KFA

KERNFORSCHUNGSANLAGE JÜLICH GmbH

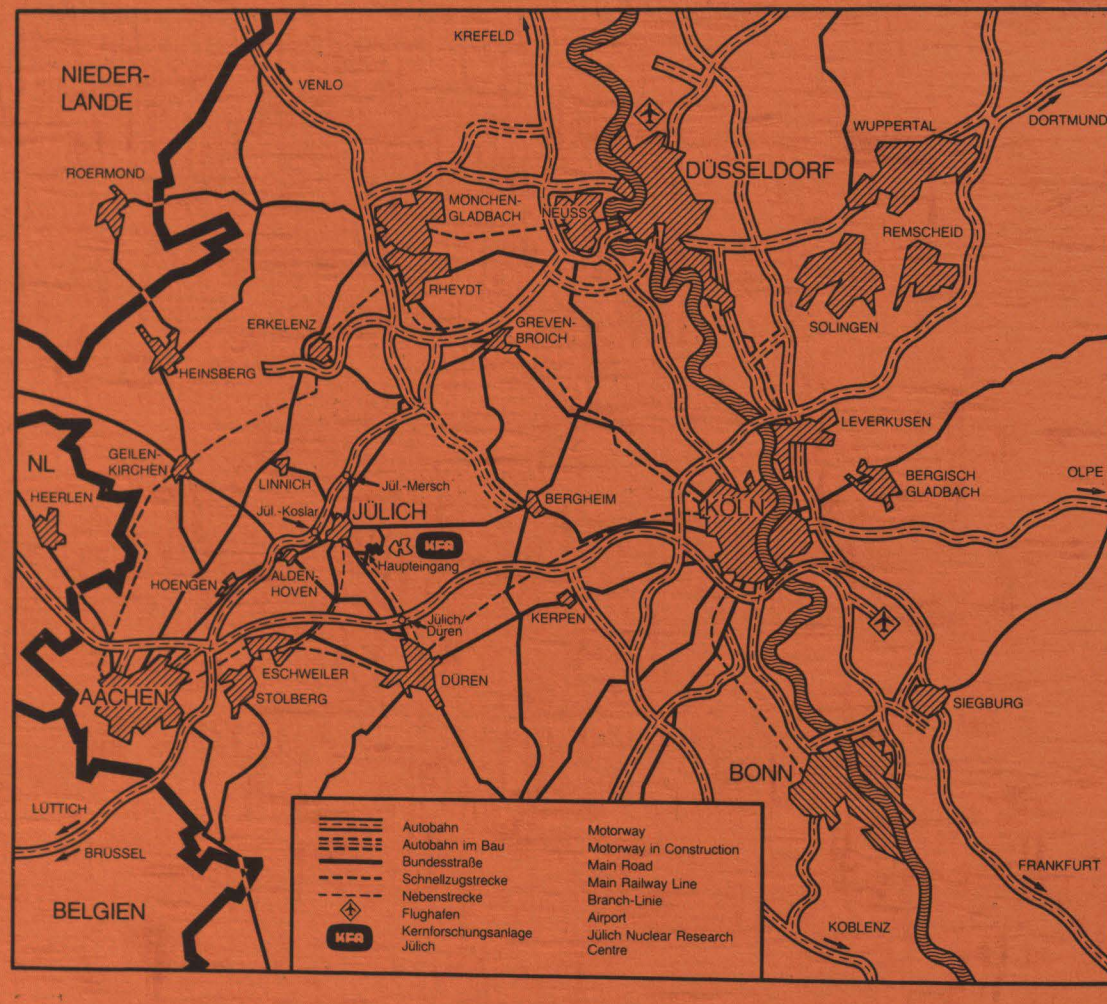
Zentrallabor für Elektronik

Multiprozessorsystem
zur Automatisierung von
Kraftstoffreglerprüfständen

von
K. Zwoll, J. Schmidt,
D. Abbott, B. Pothen

Dieser Bericht veröffentlicht u.a. Ergebnisse von Forschungsvorhaben, die vom Bundesministerium für Forschung und Technologie im Rahmen des 2. DV-Programms (Projekt · Prozeßlenkung mit DV-Anlagen, KfK-Karlsruhe) gefördert wurden.

Jül - Spez - 198
März 1983
ISSN 0343-7639



Als Manuskript gedruckt

Spezielle Berichte der Kernforschungsanlage Jülich – Nr. 198
Zentrallabor für Elektronik Jül - Spez - 198

Zu beziehen durch: ZENTRALBIBLIOTHEK der Kernforschungsanlage Jülich GmbH
Postfach 1913 · D-5170 Jülich (Bundesrepublik Deutschland)
Telefon: 02461/610 · Telex: 833556 kfa d

Multiprozessorsystem
zur Automatisierung von
Kraftstoffreglerprüfständen

von
K. Zwoll*, J. Schmidt*,
D. Abbott**, B. Pothen***

* Zentrallabor für Elektronik der KFA Jülich GmbH
** Zentrallabor für Elektronik der KFA Jülich GmbH
jetzt: Interface Standards, Fremont, CA/USA
*** Pierburg Luftfahrtgeräte-Union GmbH, Neuß (PLU)

Dieser Bericht veröffentlicht u.a. Ergebnisse von Forschungsvorhaben, die vom Bundesministerium für Forschung und Technologie im Rahmen des 2. DV-Programms (Projekt Prozeßlenkung mit DV-Anlagen, KfK-Karlsruhe) gefördert wurden.

Das beschriebene Projekt wurde im Rahmen eines Zusammenarbeitsvertrages zwischen der Firma Pierburg Luftfahrtgeräte Union GmbH, Neuß, und der Kernforschungsanlage Jülich GmbH, Zentrallabor für Elektronik, realisiert.

ZUSAMMENFASSUNG

Im Rahmen der Qualitätssicherung bei der Produktion von Flugkraftstoffreglern muß mit Hilfe eines Prüfstandes durch Aneinanderkettung einer Vielzahl statischer Prüfpunkte die ordnungsgemäße Funktion der kompletten Baugruppe nachgewiesen werden. Die Einstellung und Überwachung der Prüfparameter wie Drehzahl, Drücke, Durchflüsse und elektrische Signale erfordern einen hohen meßtechnischen Aufwand. Eine lückenlose Dokumentation ist eine unabdingbare Voraussetzung. Zur Lösung dieser Aufgabenstellung wurde ein verteilt angeordnetes Multiprozessorsystem entwickelt, das über die zentral organisierte Prozeßleitfunktion den Simultanbetrieb von bis zu 3 Prüfständen erlaubt. Auf Basis käuflicher Mikrorechnerkarten und eines eigens entwickelten Multiprozessorbetriebssystems wurden Funktionseinheiten implementiert, die über einen Botschaftsmechanismus miteinander verkehren. Eine streng anwendungsorientierte, interpretative Prüfsprache "PRAS" erlaubt eine besonders flexible und transparente Anpassung an die jeweilige Prüfaufgabe.

INHALTSVERZEICHNIS

1. EINLEITUNG

Von der Luftfahrtindustrie wird schon seit Jahrzehnten seitens der Abnehmer und der zuständigen Aufsichtsbehörden ein durchgehendes Qualitätssicherungssystem gefordert. Der Nachweis der Funktionsfähigkeit dieses Systems ist grundsätzlich Voraussetzung für den Abschluß von Entwicklungs- und Lieferverträgen. Qualitätsplanung und -steuerung sowie lückenlose Dokumentation sind in diesem Industriezweig längst geübte Praxis. Gerade wegen des vergleichsweise hohen Qualitätssicherungsaufwandes gilt es auch in der Luftfahrtindustrie, den in vielen Veröffentlichungen und Vorträgen der letzten Zeit beklagten Rückstand der Rationalisierung und Automatisierung in der Qualitätssicherung im Vergleich zu entsprechenden Entwicklungen in der Fertigung selbst, zügig auszugleichen. Dieser Rückstand hat, verstärkt durch die noch immer zunehmenden Qualitätsanforderungen, zur Folge, daß der relative Anteil des Qualitätssicherungsaufwandes an den Gesamtherstellkosten steigt und daß Kontrollzeiten die weitere Verkürzung der Fertigungsdurchlaufzeiten behindern.

Ein Beispiel für Maßnahmen auf der Ebene der Teilefertigung, die eine schrittweise Verbesserung dieser Situation zum Ziel haben, ist der Einsatz von CNC-Koordinatenmeßmaschinen. Das entscheidende an diesem Schritt ist der hohe Automatisierungsgrad, denn genaueste handbetätigte Meßmaschinen werden seit Jahren im Fertigungsbereich bereits eingesetzt. Die Aufgabe der Qualitätssicherung, den Ausstoß auch einer hochautomatisierten Fertigung zu überwachen, ist mit herkömmlichen Methoden meist nur sehr unwirtschaftlich zu erfüllen. Deutlichere Fortschritte werden in diesem Bereich erst erwartet, wenn die weitere Entwicklung die Integration von automatisierten Prüf- und Meßoperationen in den eigentlichen Bearbeitungsprozeß erlaubt /1/2/.

Das in diesem Bericht beschriebene Projekt zur Automatisierung von Kraftstoffreglerprüfständen für Strahltriebwerke mit Hilfe eines Multiprozessorsystems erweitert

konsequent die Anwendung neuer Technologien auf die Prüfung und Abnahme vollständiger Geräte und Baugruppen. Die hohen Investitionen für das komplexe Entwicklungsprojekt erfordern eine Auslegung, die auch die Übertragbarkeit auf andere Automatisierungsaufgaben und die spätere Erweiterbarkeit garantiert. Die anspruchsvolle Aufgabenstellung bot für jeden beteiligten Entwickler eine Herausforderung, neuere Technologien im Bereich der Datenverarbeitung und Elektronik in einen vornehmlich maschinenbaulich orientierten Fertigungsbetrieb zu integrieren.

2. PRÜFOBJEKT

Ein Kraftstoffregler ist wesentlicher Bestandteil des Regelsystems eines Flugzeug-Strahltriebwerkes neuester Entwicklung /3/. Bild 2-1 zeigt den Prüfstandsaufbau eines solchen Reglers.

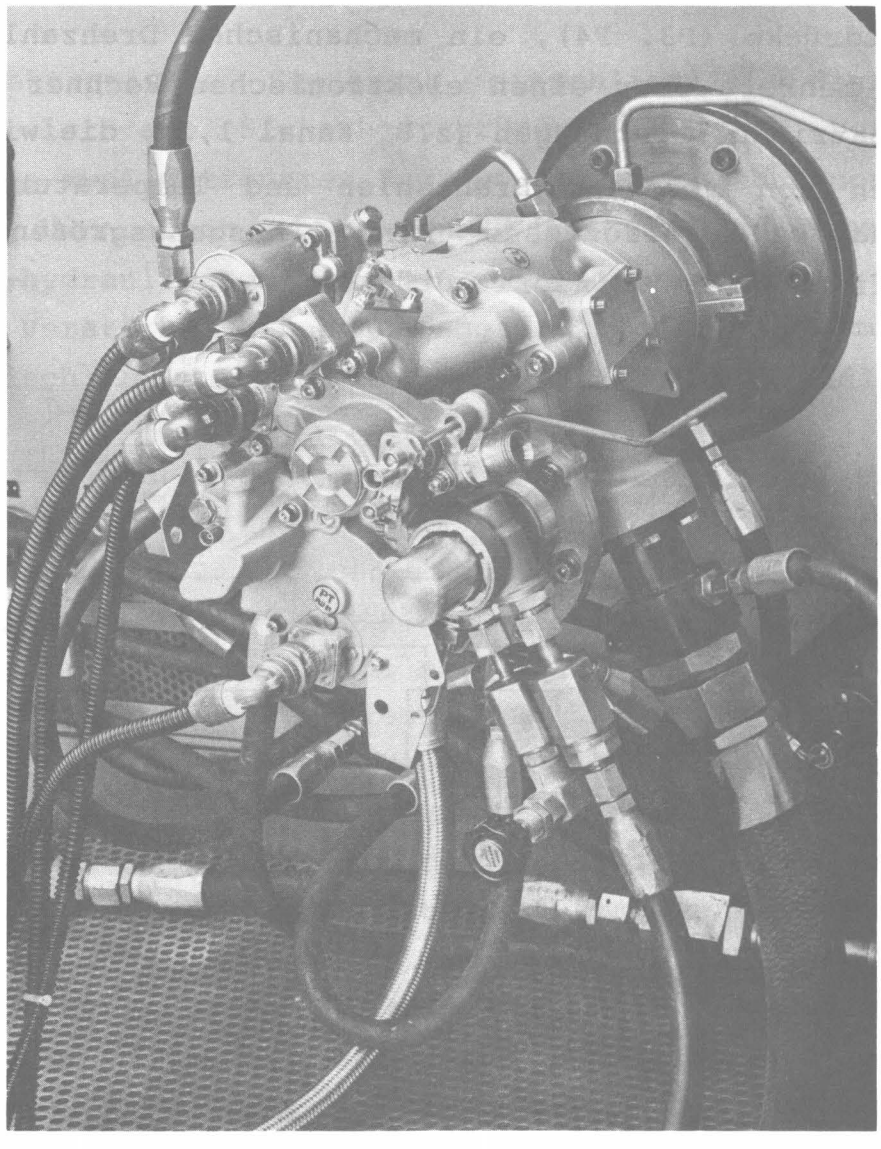


Abb. 2-1 Kraftstoffregler

Es handelt sich um ein hydromechanisches Gerät mit den Hauptaufgaben, den Flugkraftstoff mit Hilfe einer Hochleistungszahnradpumpe in Abhängigkeit von verschiedenen Triebwerksparametern auf den vor der Brennkammer benötigten Einspritzdruck zu fördern. Darüber hinaus steuert er den

Anlaß- und Abschaltvorgang des Triebwerkes und überwacht verschiedene Grenzwerte.

In Abb. 2-2 ist in schematisierter Form der Aufbau des Kraftstoffreglers dargestellt. Auf der linken Seite befindet sich der Einlaß von den Flügeltanks, auf der rechten der Auslaß zum Triebwerk. Die Eingangsgrößen sind verschiedene Luftdrücke (P3, P4), ein mechanisches Drehzahlsignal (NH) und mehrere über einen elektronischen Rechner ermittelte elektrische Spannungen (z.B. Kanal 1,2), die wiederum Funktionen von Drücken, Drehzahlen und Temperaturen des thermodynamischen Prozesses sind. Ausgangsgrößen sind Kraftstoffmengen und Einspritzdrücke.

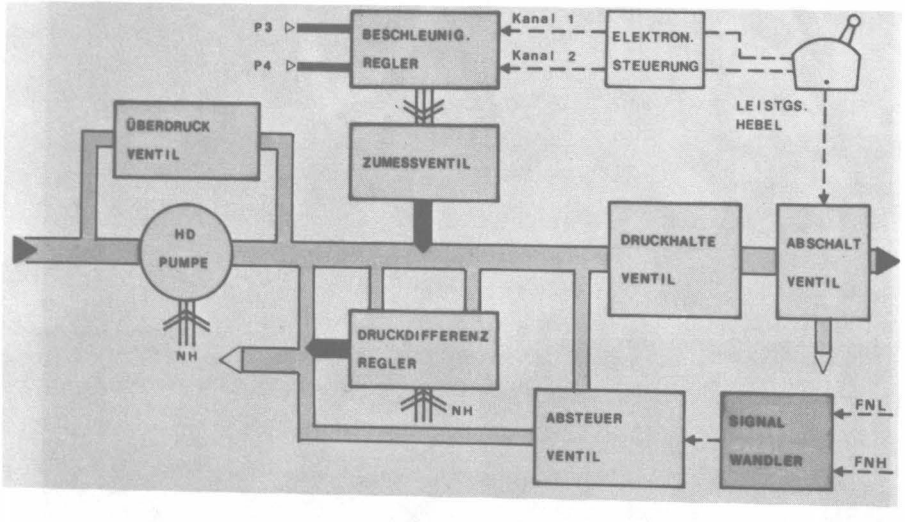


Abb. 2-2 Schema Kraftstoffregler

Die Hochdruckzahnradpumpe wird von der Welle des Hochdruckteils der Gasturbine über eine Untersetzung angetrieben (NH). Ihr der Drehzahl proportionales Fördervolumen ist von der Regelung nicht direkt beeinflußbar. Die Mengenregelung erfolgt deshalb durch Absteuerung des jeweils zuviel geförderten Anteils und dessen Rückleitung in das Kraftstoffversorgungssystem. Eine typische Aufgabe des Systems ist die Steuerung einer Triebwerksbeschleunigung auf eine höhere Drehzahl, wenn der Flugzeugführer eine höhere Leistung anfordert. Über den Leistungshebel wird eine elektri-

sche Spannung verändert, die an der elektronischen Steuerung anliegt. Unter Berücksichtigung verschiedener Prozeßgrößen, deren Eingänge im Blockschaltbild nicht dargestellt sind, bildet der in der Steuerung enthaltene Rechner ein Ausgangssignal, das wiederum in Form einer elektrischen Stromänderung an den Beschleunigungsregler weitergegeben wird.

Die elektronische Steuerung einschließlich Signalübertragung ist aus Sicherheitsgründen zweifach vorhanden, daher die zwei getrennten Kanäle. Der Beschleunigungsregler erzeugt über einen elektro-pneumatischen und einen pneumatisch-hydraulischen Wandler mit nachgeschaltetem hydraulischem Verstärker einen Ausgangsdruck, der das Zumeßventil hydraulisch verstellt. Die zweimalige Signalwandlung ist notwendig, weil in der Zwischenstufe das Verhältnis der beiden für den augenblicklichen Betriebszustand des Verdichters charakteristischen Drücke P4/P3 berücksichtigt werden muß. Die Durchflußquerschnittsänderung im Zumeßventil als Funktion des Verstellweges wird über ein kalibriertes Profil erzeugt. Der der Brennkammer zur Temperaturerhöhung und damit zur Beschleunigung des Triebwerks zuzuführende Brennstoffüberschuß ist außerdem durch die momentane Drehzahl der Welle des Hochdruckteils begrenzt. Um diese Abhängigkeit zu berücksichtigen, wird der Druckabfall über dem eingestellten Querschnitt des Zumeßventils mit Hilfe des Druckdifferenzreglers dem Quadrat dieser Drehzahl proportional gehalten. Nach dem Durchflußgesetz einer Blende wird damit die durch den vorgegebenen Querschnitt strömende Brennstoffmenge proportional zur Triebwerkdrehzahl.

Auf der Basis des vereinfachten Blockschaltbildes (Abb. 2-2) zeigt Bild 2-3 die Stell- und Meßgrößen des Reglers. Dies sind, wie oben erwähnt, im wesentlichen: Steuerströme, Drehzahlen, Luft- und Kraftstoffdrücke. Die Eingangsgrößen werden dabei von außen mit Hilfe der beschriebenen Signale zugeführt, die für den Prüfvorgang benötigten Meßgrößen werden an verschiedenen Baugruppen

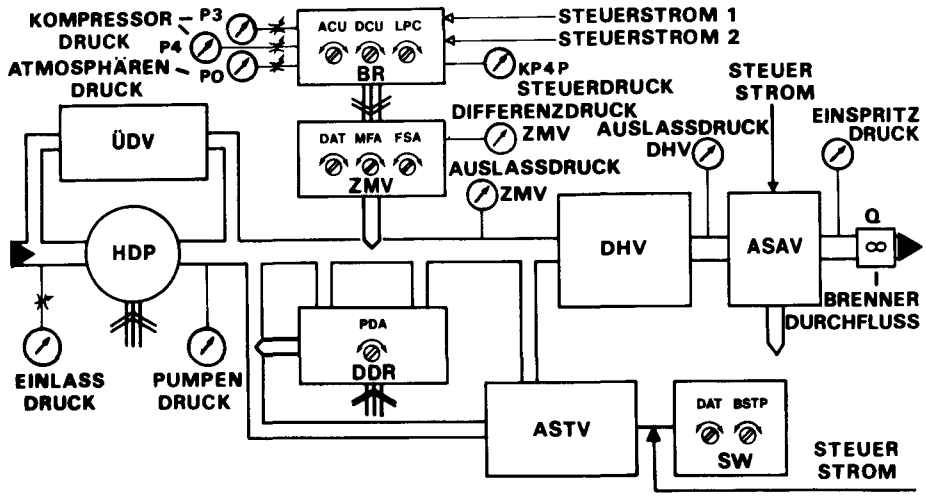


Abb. 2-3 Stell- und Meßgrößen am Kraftstoffregler

erfaßt. Mehrere Parameter sind über mechanische Kalibriereinrichtungen justierbar (vgl. Tabelle 4-1).

3. PRÜFAUFGABE

Der Endabnahme der Geräte im Prüffeld kommt eine ganz wesentliche Bedeutung zu. Der Abnahme-Prüfablauf dient zum Nachweis der vertraglich zugesicherten Eigenschaften an jedem fertigen Gerät. Zur Endabnahme der Geräte gehören als Bestandteil der Qualitätssicherung zusätzliche Testzyklen als Nachweis der Zuverlässigkeit und der Reproduzierbarkeit aller Funktionen. Damit sollen auch die Fehler noch abgefangen werden, die sich trotz aller intensiven Qualitätssicherungsmaßnahmen in den verschiedenen Fertigungsstufen einschleichen können. Dazu gehören auch Tests mit erhöhter Prüfbelastung. Der Endabnahme sind umfangreiche Kalibrierarbeiten an den oben erwähnten Einstellvorrichtungen vorgelagert, um zu einem vom Auftraggeber abnahmefähigen Serienprodukt zu gelangen.

Die Geräte kommen im Laufe ihrer Nutzungszeit im allgemeinen mehrmals zur Instandsetzung und periodischen Grundüberholung zum Hersteller zurück. Im Rahmen dieser Bearbeitungen sind die gleichen Prüfungen und Abnahmen zu wiederholen, wobei der Rückgriff auf die ursprüngliche Dokumentation wichtige Hinweise auf inzwischen eingetretene Veränderungen gibt. Aus diesem Grunde ist die Gewinnung von dokumentationsfähigen Daten ein unerläßlicher Teil der Prüfaufgaben.

4. KONVENTIONELLER PRÜFABLAUF

Die skizzierten Einstellarbeiten, die Endabnahme und die dabei gewonnenen Prüfdaten erfordern einen komplexen Prüfablauf. Da eine dynamische Simulation der Betriebszustände des Triebwerks einen wirtschaftlich nicht vertretbaren Aufwand verursachen würde, beschränkt man sich auf die

Blatt: 3 von 11 | Benennung: FLOW CONTROL - ENGINE FUEL | Serien-Nr.:

5.0 V.M.O. HOT SOAK AND CYCLIC ENDURANCE TEST

DREHZAHL min^{-1}	KP4P PSIA	DIFFERENZDRUCK PSI VMO	DURCHFLUSS CPH	
8115	100		45	
8116	U-110	50x WECHSELN 5x PRO MINUTE		
7710	20	117	A±15	170.4
7710	30	116	B±15	276
7710	40	115	C±15	395
7710	50	118	D±15	505
7710	60	118	E±15	616
7710	70	117	F±15	728

* 5 Minuten bei dieser Einstellung
U= UMGEBUNGSDRUCK

6.2 L.P.C. INITIATION AND RATE OF PULLDOWN LANE 1

DREHZAHL min^{-1}	7710	P4 PSIA	280,0	P3 PSIA	75,7

DURCHFLUSS GPH			STROM mA
896	925	950	0
710	727	730	350
896	923	950	100
	923		0
A-5	922	A+5	100
B-30	920	B	180
	729		350
C-85	658	C-55	400
C-155	603	C-125	450
458	474	486	900

▽ = ERHÖHEN
△ = VERRINGERN
A ACU = 140 mA
B DCU = 900-1000 mA
C

6.3 HYSTERESIS LANE 1

STROM mA ▽	DURCHFLUSS GPH	△ STROM max.30 mA	DURCHFLUSS GPH	STROM mA △
0	952		952	0
340	800	25	800	315
430	700	24	700	406
503	600	20	600	483
900	547		547	900

Änd. Nr. 1 2 3
Datum 3.11.81 Name Becker Bearb.
Datum 3.11.81 Name Geprüft
PLU-Formblatt 163

Abb. 4-1 Beispiel eines Prüfprotokollausschnitts mit den Prüfschritten 5.0, 6.2 und 6.3

Aneinanderreihung einer Vielzahl statischer Prüfpunkte, deren Folge in einer verbindlichen Prüfvorschrift festgelegt ist. Diese soll einen systematischen, gleichbleibenden Prüfablauf mit Test und Kalibrierung des Kraftstoffreglers garantieren. Funktionell zusammengehörige Prüfsequenzen sind in möglichst unabhängigen Prüfschritten zusammengefaßt. Die Prüfvorschrift enthält Eingabefelder, die während des Prüfablaufs vom Bediener mit den von den Meßinstrumenten angezeigten Werten ausgefüllt werden. Auf diese Weise entsteht ein Prüfprotokoll, das archiviert und über die gesamte Lebensdauer des Prüflings verfügbar gehalten wird.

	STELLGRÖSSEN		MESSGRÖSSEN	
PRÜFSTAND	1. Drehzahl	NH	1. Drehzahl	NH
	2. Kompressordruck	P4	2. Kompressordruck	P4
	3. Kompressordruck	P3	3. Kompressordruck	P3
	4. Atmosphärendruck	PØ	4. Atmosphärendruck	PØ
	5. Einlaßdruck	PBOOST	5. Einlaßdruck	PBOOST
	6. Steuerstrom	IXLPC	6. Steuerstrom	IXLPC
	7. Regelventil	VX	7. Regelventil	VX
	8. Regelventil	VY	8. Regelventil	VY
	9. Umschalter	Kanal 1/2	9. Umschalter	Kanal 1/2
	10. Umschalter	MLPC/ELPC	10. Umschalter	MLPC/ELPC
KRAFTSTOFFREGLER	1. DDR-Einsteller	PDA	1. Einlaßdruck	P/BOOST
	2. ZMV-Bezugspunkt-Einsteller	DAT	2. Pumpendruck	P/PUMP
	3. ZMV.Max.Fluß-Einsteller	MFA	3. Einspritzdruck	P/BURNER
	4. ZMV-Begrenz.Einsteller	FSA	4. ZMV-Auslaßdruck	P/DS-VMO
	5. BR-Beschleunig.-Einst.	ACU	5. ZMV-Differenzdruck	P/D-VMO
	6. BR-Verzögerungs-Einst.	DCU	6. DHV-Auslaßdruck	P/D-PRV
	7. BR-Bezugspunkt-Einst.	LPC	7. Brenner Durchfluß	Q/BURNER
	8. SW-Bezugspunkt-Einst.	OAT		
	9. SW-Begrenzungs-Einst.	BSTP		

Tab. 4-1 Meß- und Stellgrößen Prüfstand, Kraftstoffregler (vgl. Abb. 2-3)

Die für den Prüfablauf benötigten Prüfparameter werden mit Hilfe eines dafür konstruierten Prüfstandes generiert. Für die Grundkalibrierung sind etwa 50 verschiedene Funktionen

und Kennlinien zu durchfahren und einzuregulieren. Je Funktion werden 3 bis 12 Prüfpunkte mit unterschiedlichen Eingangsgrößen und Parameterkombinationen eingestellt. 15 verschiedene Stell- und Meßgrößen werden überwacht. Tabelle 4-1 zeigt eine Auflistung der Meß- und Stellgrößen des Prüfstandes und des Kraftstoffreglers.

Um in einem iterativen Ablauf zu einer optimalen Einstellung zu gelangen, müssen die Teilfunktionen mehrmals nacheinander durchlaufen werden. Während eines Prüf- und Abnahmeablaufs werden bis zu 300 Parameter-Einstelloperationen durchgeführt und etwa ebenso viele Meßwerte abgelesen und notiert.

Nach Abschluß der Einstellarbeiten folgt der eigentliche Abnahmelauf, bei dem alle Funktionen des Prüflings in der vorgeschriebenen Reihenfolge ohne Unterbrechung abgefahren werden. Einige ausgewählte Teilfunktionen werden anschließend aus Sicherheitsgründen im Beisein eines Mitarbeiters der Qualitätskontrolle wiederholt.

4.1 Prüfstand

Die Abb. 4.1-1 zeigt einen konventionellen Prüfstand, wie er zur Durchführung der Prüfaufgaben und -abläufe eingesetzt wird. Zur Simulation der Triebwerkeinflußgrößen benötigt man eine Reihe von Zusatzeinrichtungen. Dazu gehören neben Vordruckpumpen, Kompressoren und Vakuumpumpen ein Gleichstrommotor zum Antrieb der im Prüfling integrierten Hochdruckzahnradpumpe. Darüber hinaus werden Signalgeber, die Parameter der elektronischen Steuerung simulieren und ein System zur Konstantregelung der Temperatur des Prüfmediums (hiervon hängt die Genauigkeit der Mengenmessung ab), verwandt.

Das Prüfmedium ist Flugkraftstoff. Das Prüffeld ist daher ein explosionsgefährdeter Bereich. Alle Prüf- und Meßeinrichtungen sowie deren Installation müssen den Sicherheits-

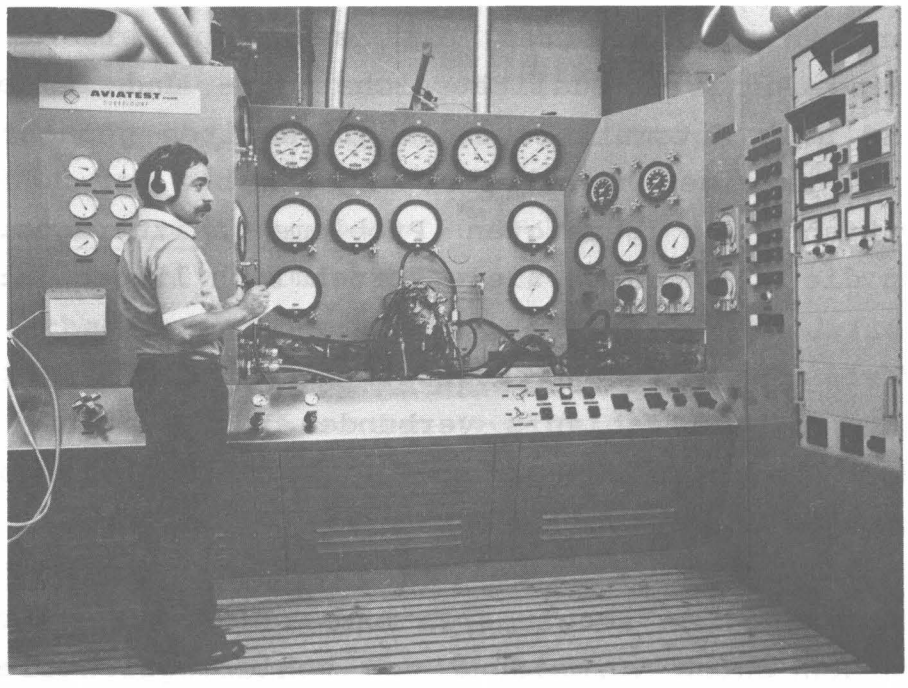


Abb. 4.1-1 Konventioneller Prüfstand

vorschriften entsprechen /4/. Hohe Drehzahlen und Strömungsgeschwindigkeiten im Prüfsystem erzeugen einen Lärmpegel, der das ständige Tragen eines Gehörschutzes erfordert.

Die Belastung des Prüfpersonals ist aufgrund der hohen Anforderungen an Zuverlässigkeit und Verantwortungsbewußtsein sehr hoch. Lärmbelästigungen und andere Umgebungsbedingungen kommen erschwerend hinzu. Wegen des hohen Investitionswertes der Prüfstandanlage ist im Sinne einer wirtschaftlichen Auslastung im Mehrschichtbetrieb zu fahren.

Die auf herkömmliche Art manuell erstellten und archivierten Prüfprotokolle sind einer systematischen Auswertung, aus der wichtige Informationen für die Weiterentwicklung der Erzeugnisse, der Fertigungsverfahren und der Qualitätssicherungsmethoden gewonnen werden könnten, nur schwer zugänglich. Der dazu erforderliche manuelle Verwaltungsaufwand ist wirtschaftlich kaum zu vertreten.

5. ZIELE DES AUTOMATISIERUNGSVORHABENS

Aus den wenigen zitierten Hinweisen ist ersichtlich, daß für diese Art von Qualitätssicherungs- und Prüfaufgaben dringend nach Automatisierungsmöglichkeiten gesucht werden mußte, um zu einer besseren Lösung der bei der Schilderung des Prüfablaufes gezeigten Probleme beizutragen. Dabei sollten die gewünschten Ziele vornehmlich liegen in:

- Verbesserung der Wirtschaftlichkeit durch Beschleunigung des Prüfablaufs verbunden mit erhöhter Nutzung der mit hohen Kosten beaufschlagten Anlage und Reduzierung des manuellen Aufwandes.

- Minderung von scheinbaren und echten Qualitätsschwankungen durch Vermeidung von Fehlermöglichkeiten beim Ablesen und Übertragen der Meßwerte in das Prüfprotokoll.

- Verbesserung der Arbeitsbedingungen im Prüffeld.

Die geplante Automatisierung sollte sich zunächst auf den Prüfstand selbst beziehen. Dabei sollen alle Meßgrößen rechnerunterstützt erfaßt, vollständige Abnahmeläufe, das Durchfahren bestimmter Teilfunktionen und das Anwählen bestimmter Prüfpunkte von einem Bedienpult ausgelöst werden können. Die Einstellarbeiten am Kraftstoffregler werden weiterhin von einem Bedienungsmann vorgenommen.

Die zunächst denkbare Automatisierung der Einstellarbeiten mit Hilfe geeigneter Stellantriebe läßt sich ohne sehr großen Aufwand nicht erreichen und wurde deshalb verworfen, zumal die Grundkonstruktion des Kraftstoffreglers in mehreren Punkten für die Adaption von geeigneten Stellgliedern hätte ausgelegt sein müssen.

5.1 Anforderungen an das Automatisierungssystem

Zur Erreichung der oben global formulierten Ziele lassen sich die notwendigen Systemeigenschaften festlegen, die das Automatisierungssystem auszeichnen sollten:

Setzen von Prüfbedingungen

Alle Testbedingungen müssen in einer reproduzierbaren Sequenz eingestellt, verifiziert und gegen Toleranzüberschreitungen abgeprüft werden (zum Teil mit Hilfe geschlossener Regelkreise).

Lesen, Anzeigen, Speichern und Ausgeben von Meßwerten

Das Auslesen, Anzeigen von Meßwerten und Prüfen gegen Grenzwerte, die durch Testvorschriften festgelegt sind, ebenso die Archivierung der Meßwerte und das Protokollieren in einem vorgeschriebenen Format als Prüfprotokoll (Test Schedule) soll möglich sein. An die Ausgabe des Prüfprotokolls werden besondere Forderungen gestellt. Denn die im Rahmen eines automatisierten Prüfablaufs gewonnenen Prüfprotokolle sollten mit denen der bis jetzt konventionell erstellten in Format und Aussehen weitgehend kompatibel sein, um eine gleichbleibende, lückenlose Dokumentation über die gesamte Lebenszeit des Produktes liefern zu können.

Bedienerführung im Dialog

Ausgaben von Bedienerhinweisen und auch manuelle Eingabe von nicht on-line erfaßbaren Prüfdaten.

Schutz des Prüflings- und Prüfstandes

Fehleingaben des Bedienungspersonals und Hardwarefehler dürfen nicht zur Beschädigung der Komponenten führen.

Test Flexibilität

Der Ablauf der Prüfung kann vom Bedienungsmann in mehrfacher Weise beeinflußt werden. Zusammenhängende Prüfschritte können zu jeder Zeit unterbrochen werden, beliebige andere angewählt oder Einzelwerte gesetzt werden.

Echtzeitdarstellung der Meßwerte

Alle digitalen und analogen Meßwerte sollen möglichst transparent und übersichtlich dem Operator in komprimierter Form auf einem Sichtgerät dargestellt werden.

Programmierbarkeit des Prüfablaufes

Der Prüfablauf soll generell in einer für den Bediener verständlichen, leicht zu erlernenden, applikationsorientierten Sprache beschreibbar und damit auch mit wenig Aufwand änderbar sein.

Betriebsprotokollausgabe

Der zeitliche Prüfablauf soll auf einem zentralen Drucker protokolliert werden.

Kalibriermöglichkeit für Meßwertaufnehmer

Die Unlinearitäten und Langzeitdriften der Meßwertaufnehmer sollen jederzeit über bedienerfreundliche Prozeduren korrigierbar sein.

Graceful Degradation

Bei Ausfall von Systemkomponenten soll der Prüfablauf in einer degenerierten Betriebsart weiterhin möglich sein, wobei die Qualität der Prüfung möglichst beibehalten werden sollte, die Produktivität jedoch sinken kann.

Servicefreundlichkeit

Eine modulare Auslegung des Systems, verbunden mit on-line Testhilfen sollte eine geringstmögliche Reparaturzeit garantieren.

Flexibilität und Ausbaufähigkeit

Das Automatisierungssystem sollte so ausgelegt werden, daß es in wesentlichen Teilen ausbaufähig und auf vergleichbare Aufgaben übertragbar ist. Deshalb werden besondere Anforderungen an die Flexibilität und den modularen Ausbau des Systems gestellt, um auch die Prüfstände mit vertretbarem Aufwand und mit möglichst firmeneigenen Mitteln auf Änderungen der Prüfabläufe und Prüfinhalte umstellen zu können.

Integrierbarkeit in ein vorhandenes Produktionssteuerungssystem

Das geplante System sollte einen weitgehend dezentralen, autarken Prüfablauf ermöglichen und zu einem späteren Zeitpunkt in das vorhandene und im weiteren Ausbau befindliche rechnerunterstützte Produktionssteuerungssystem einbezogen werden.

5.1 Betriebsarten des rechnergesteuerten Prüfstandes

Aus den dargestellten Eigenschaften lassen sich mit Hilfe der bereits an den Prüfständen installierten Komponenten die folgenden 3 Betriebsarten ableiten:

- Manueller Betrieb
  In dieser Betriebsart wird der Prüfstand in konventioneller Art gefahren, d.h. das Setzen, Ablesen und Protokollieren von Prüfwerten läuft unter alleiniger Kontrolle des Operators ab. Alle Meßwerte werden von den im Prüfstand installierten analogen Meßsystemen abgelesen.

- Semi-Automatik Betrieb
  Die Einstellung der Vorgabewerte erfolgt über geschlossene Regelkreise mit Toleranzüberwachung. Die Werte werdcn per Zifferntastatur eingegeben. Die Anwahl von wiederkehrenden Prüfbedingungen mit einer Gruppe von gleichen Vorgabewerten kann über Funktionstasten bzw. Codeworte eingegeben werden (Prüfkonditionseingabe mit Hilfe von Prüfmakros). Die Meßwerte werden numerisch und quasi analog auf einem Sichtgerät angezeigt, die Protokollierung geschieht manuell.

- Automatik Betrieb
  Ein Prüfprogramm übernimmt hier die Kontrolle über den Prüfablauf. Der Bedienungsmann wählt die gewünschte Prüfschrittnummer an, der Prüfablauf geschieht nach Möglichkeit vollautomatisch. Die Integration des Bedienungsmanns in den Prüfablauf geschieht per Bedienerführung. Manuelle Eingaben werden auf Plausibilität untersucht. Alle Meßwerte werden auf einem Sichtgerät dargestellt, die Archivierung der Prüfdaten wird blockweise pro Prüfschritt auf einem Massenspeicher vorgenommen. Von dort können sie sowohl dem Bediener jederzeit angezeigt werden als auch auf einem graphischen Drucker in einem programmierbaren Format dargestellt werden.

6. FUNKTIONSORIENTIERTER MEHRPROZESSOR-SYSTEMENTWURF

Aufgrund des Anforderungsprofils lassen sich zur Lösung der Automatisierungsaufgabe eine Reihe von anwendungsorientierten Funktionseinheiten definieren, die jeweils für sich abgeschlossene Aufgabenbereiche bearbeiten und deshalb logisch separierbar sind. Die Funktionseinheiten lassen sich nach Abb. 6-1 hierarchisch strukturieren, wobei jeder Funktionsblock dem entsprechenden übergeordneten seine Dienste zur Verfügung stellt.

Die organisatorische Leitfunktion wird vom Supervisor bereitgestellt; ihm obliegen die folgenden Einzelfunktionen, die er mit Hilfe der angeschlossenen peripheren Geräte, wie Massenspeicher, Protokolldrucker, Terminal und graphischem Drucker ausführen kann:

- Erstellung und Ändern von Indexdateien (Kennsätze) für die Meßdaten eines Prüfprotokolls mit den wichtigsten Prüfbedingungen und organisatorischen Daten.

- Prüfprogrammerstellung und Speicherung mit einem anwendungsbezogenen Editor

- Meßdatenarchivierung

- Betriebsprotokollgenerierung mit Ausgabe der wichtigsten Betriebsdaten und Störungsmeldungen mit Ort und Zeitpunkt des Entstehens

- Prüfprotokollerstellung mit dem Inhalt der Indexdatei und der Prüfdaten

- Spätere Meßdatenverarbeitung

Der technische Prüfablauf wird von der Ablaufsteuerung koordiniert und gesteuert. Im einzelnen ergeben sich dabei die folgenden Aufgaben:

- Initialisierung des Systems nach dem Einschalten

- Laden von Prüfprogrammen vom Supervisor

- Interpretation des Prüfprogramms im Automatikbetrieb oder einzelner Prüfparameter, die in der Semi-Automatik Betriebsart von der Bedienfeldsteuerung übermittelt werden

- Koordination der Aktivitäten aller anderen prozeßorientierten Teilfunktionen, wie z.B. DDC-Regler und Bedienfeldsteuerung

- Überwachung des Prüfablaufs auf Ausnahmebedingungen, wie z.B. Grenzwertüberschreitungen

- Aufbereitung der Prüfdaten auf eine einheitliche archivierte Form und Übergabe an den Supervisor

- Überführung des Prüfstandes in einen definierten sicheren Zustand (STANDBY) auf Anforderung des Bedieners

- Kalibrierung der Meßwertaufnehmer

Die Bedienfeldsteuerung stellt über den zugehörigen Monitor mit Tastatur die Verbindung zum Prüfstandsbediener her; dabei sind die nachstehenden Funktionen zu erfüllen:

- Starten, Unterbrechen und Beenden des automatischen Prüfablaufs

- Auslösen der Semi-Automatik-Betriebsart und Übergabe der Prüfparameter an die Ablaufsteuerung

- Auslösen der Standby-Funktion der Ablaufsteuerung

- Anzeigen von Prozeßgrößen

- Darstellen von Prüfdaten (auch aus vorhergehenden Prüfschritten)

- Unterstützung der Kalibrierfunktion

Eine weitere Funktionseinheit bildet der Mehrfach-DDC-Regler mit der Hauptaufgabe:

- Einstellen und Überwachen der Prüfbedingungen mit Hilfe digitaler Regelkreise (12 Regelkreise mit maximaler Abtastrate von 10 Hz) nach P, PI und PID Algorithmen.

Um die Arbeitsbedingungen des Kraftstoffreglers möglichst praxisnah zu gestalten, müssen einige Regelkreise auftrennbar sein und der letzte Stellwert eingefroren werden. Darüber hinaus müssen einzelne Regler auch als Folgeregelkreise verschaltet werden können (z.B. Q Burner als Funktion der Drehzahl NH).

Die Analog-Eingabe dient dem DDC-Regler als Eingabekanal. Dazu nimmt er folgende Teilfunktionen wahr:

- Erfassung von bis zu 12 analogen Prozeßmeßwerten

- Digitale Glättung mit vorgegebenen Parametern (Gleitende Mittelwertbildung)

- Offset Kompensation

- Linearisierung von Meßwertaufnehmerkennlinien

Die Analog-Ausgabe dient dem DDC-Regler als Ausgabekanal und stellt analoge Signale zur Ansteuerung der Stellglieder des Prüfstandes und unterlagerter Elektronikkomponenten zur Verfügung.

Zur Ein/Ausgabe von digitalen Prozeßdaten sind zwei weitere Funktionseinheiten vorgesehen. Diese betätigen Ventile,

überwachen sie und erfassen Meßwerte, die als BCD codierte Zahl vorliegen. Darüber hinaus wird die Integration eines frequenzproportionalen Ausgangssignals eines Drehzahlgebers vorgenommen.

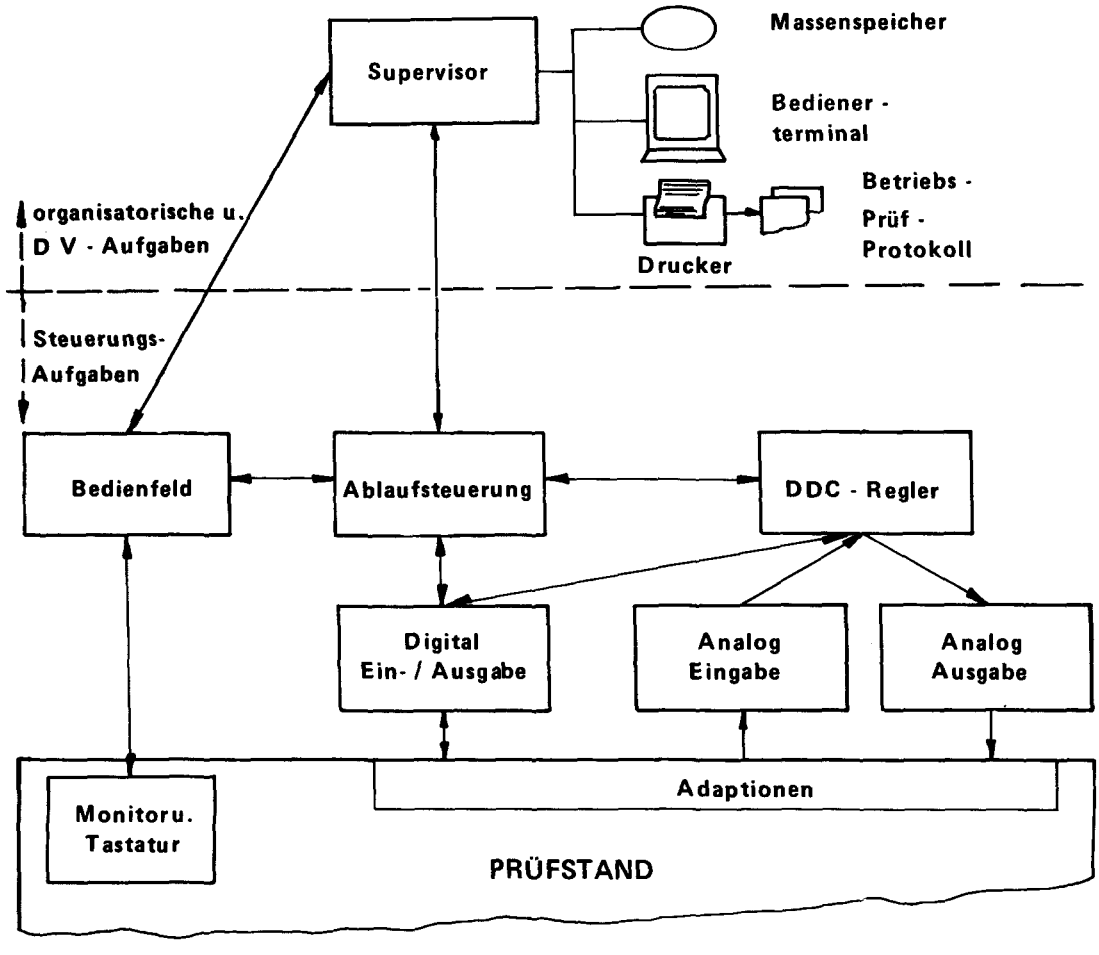


Abb. 6-1 Funktionseinheiten des Automatisierungssystems

Die geringe Auslastung der Supervisorfunktion während der Prüfabläufe führte zu der Überlegung, diese als zentrale Datenverarbeitungseinheit simultan für mehrere Prüfstände einzusetzen, um so die aufwendige Peripherie in wirtschaftlicher Weise zu nutzen. Dadurch ergibt sich eine Systemauslegung nach Abb. 6-2, wobei ein zentrales Rechnersystem mit der Supervisorfunktion drei Prüfstandssteuereinheiten bedient, die jeweils für sich die oben beschriebenen Funktionen ausführen.

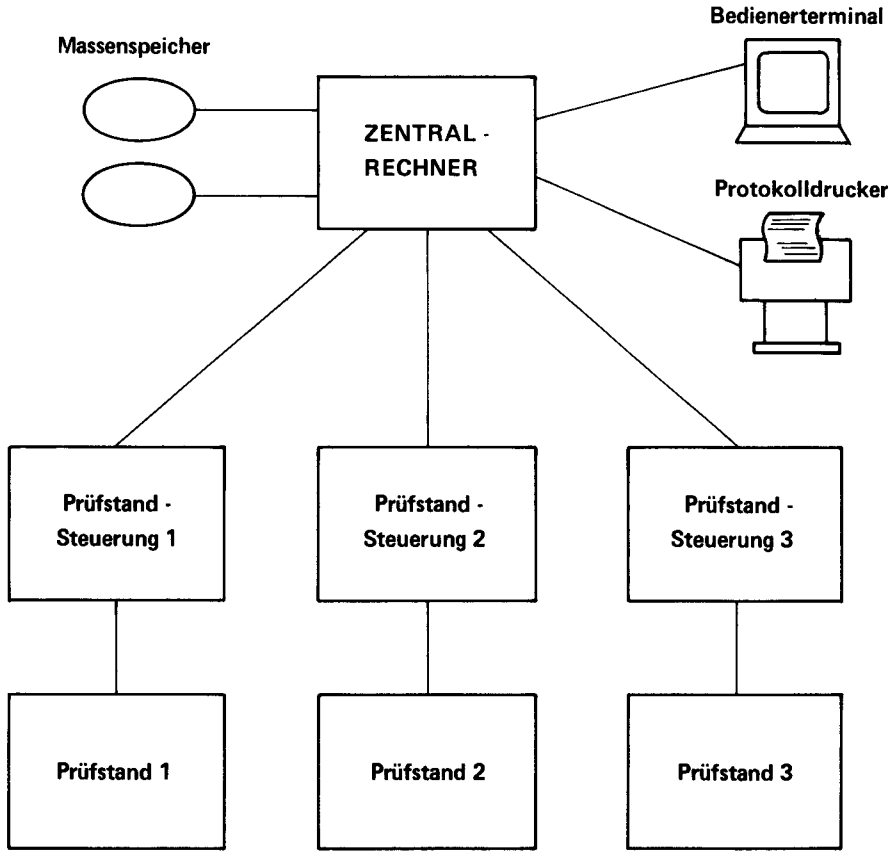


Abb. 6-2 Prüfstandsautomatisierungssystem mit zentraler Leitfunktion und dezentralen, teilautarken Prüfstandssteuereinheiten

Durch die Zentralisierung der Leitfunktion wird dieser Anlagenteil bestimmend für die Ausfallsicherheit des Gesamtsystems. Deshalb wurde bei der Konzipierung des Systems davon ausgegangen, daß selbst ein Totalausfall des zentralen Rechners nur zu einer Degenerierung des Systems bis zum Semi-Automatik-Betrieb führen soll (vgl. Kap. 11.1.1.3). Dies setzt die Autarkie der dezentralen Prüfstandssteuereinheiten voraus, soweit die steuerungsspezifischen Funktionen betrachtet werden.

Für die Realisierung des in Abb. 6-2 gezeigten Systems wurde ein Verbund von µ-Rechnern gewählt, die jeweils die Aufgaben einer der dargestellten Funktionseinheiten übernehmen können. Die einzelnen Mikrorechner sind dabei nach unterschiedlichen Verfahren gekoppelt. Während die "Rechner-Cluster" im zentralen Supervisor und in den Prüfstandssteuereinheiten aus Geschwindigkeitsgründen über parallele Bussysteme miteinander verbunden sind, wird die größere Entfernung zum zentralen System mittels langsamer serieller Datenübertragungsmethode überbrückt. Die Kommunikation der einzelnen beschriebenen Funktionseinheiten untereinander geschieht über anwendungsspezifische Botschaften auf einem Hochniveau, das von dem entwickelten Betriebssystem (vgl. Kap. 10) bereitgestellt wird.

Die wesentlichen Gründe zum Einsatz eines Multiprozessorsystems waren:

- Einplatinenrechner werden auf dem Markt von verschiedenen Herstellern und in einer großen Anzahl von Versionen zu relativ niedrigen Preisen angeboten.

- Die dadurch mögliche funktionsbezogene Nutzung jedes Prozessors schafft durch Festlegung geeigneter Schnittstellen eine sehr große Unabhängigkeit der einzelnen Module untereinander. Wenn sich die Anforderungen an einen Modul ändern, muß nur dieser modifiziert werden, die anderen können unverändert bleiben.

- Die exakt definierbaren Schnittstellen der Funktionsmodule untereinander findet ihre Entsprechung in einer klaren Projektstruktur, in deren Rahmen kleine unabhängige Entwicklungsgruppen auch in einem loseren Verbund zu effizienten Ergebnissen kommen können. Dieser Grund war sehr wesentlich bei dem hier beschriebenen Entwicklungsvorhaben, weil die Implementatoren über längere Zeiträume getrennt voneinander arbeiten mußten.

- Die Leistungsfähigkeit des Gesamtsystems kann durch die mögliche Parallelarbeit der einzelnen µ-Rechner gesteigert werden; hier ist insbesondere der dauernd erforderliche Eingriff der DDC-Regler zu nennen.

- Ein Multiprozessorsystem verfügt über eine sehr feinkörnige Ausbaufähigkeit und Flexibilität; dadurch können weitere - zunächst nicht vorgesehene - Funktionen auch nachträglich ohne großen Aufwand in das System integriert werden.

- Verteilte Rechnerleistung kann die Systemverfügbarkeit erhöhen, ein Fehler in einem Teilsystem muß nicht notwendigerweise das Gesamtsystem blockieren.

- Verglichen mit dem zentralen Bus eines herkömmlichen Minirechners werden durch den möglichen funktionsorientierten Systemlayout im Rahmen eines Mehrprozessorsystems die Anforderung an die Bandbreite des Bussystems gesenkt.

7. AUSWAHL DES MEHRPROZESSORSYSTEMS

Um den verfügbaren Mittelrahmen nicht zu sprengen, mußte bei den Entwicklungsarbeiten möglichst auf käufliche OEM-Produkte zurückgegriffen werden. Da das Angebot an Multiprozessorsystemen mit ausreichendem Funktionsspektrum zum Projektstart gering, darüber hinaus die geforderte Leistungsfähigkeit sowohl hinsichtlich steuerungs-, als auch DV-spezifischer Aufgaben hoch war, fokussierten sich die Untersuchungen auf die im folgenden dargestellten Mikrorechnersysteme. Deren jeweilige Vorteile konnten im Rahmen eines verteilten Systems unter Führung eines einheitlichen Betriebssystems (vgl. Kap. 10) genutzt werden.

7.1 MULTIBUS Mikrorechnersystem

Das Intel iSBC-Mikrorechnersystem /5/ basiert auf der Grundlage der MULTIBUS-Architektur /6/, die vom IEEE als Standard (Nummer 796) zum Aufbau von Ein- und Mehrrechnersystemen festgelegt worden ist. Im Jahre 1976 wurde der MULTIBUS-Systembus als flexible Schnittstelle für die 8 Bit-Familie (iSBC-80) von Mikrorechnern von der Firma Intel vorgestellt. Als die Produkte des 16 Bit-Einplatinenrechners im Jahre 1978 eingeführt wurden, waren diese von Anfang an MULTIBUS-kompatibel. Im weiteren Verlauf der Entwicklung entstand auf der Grundlage des Systembusses ein einheitliches Platinensystem, das Einplatinenrechner, Erweiterungskarten für Speicher, digitale, analoge Ein/Ausgabe- und Peripheriesteuergeräte umfaßte. In den USA hat sich darüber hinaus ein breiter Markt von OEM-Zulieferern gebildet.

Der MULTIBUS-Systembus unterstützt die direkte Adressierung von bis zu einem Megabyte und den parallelen Datentransfer von 8 und 16 Bit Breite. Die Struktur ist auf dem Master-Slave-Konzept aufgebaut, wobei der Master-Einheit eines Systems die Steuerung der MULTIBUS-Schnittstelle zugeordnet ist und die Slave-Baugruppe auf die vom Master übermittel-

ten Befehle reagiert. Dabei läuft die Steuerung im Quittungsbetrieb ab, so daß Einheiten mit unterschiedlichen Geschwindigkeiten über den Systembus verkehren können. Eine Datenübertragungsrate von bis 5 Millionen Worte pro Sekunde ist möglich.

Grundsätzlich können an einem Systembus bis zu 16 Master-Baugruppen - mit internen Speichern ausgerüstet - simultan arbeiten. Die Zuteilung des Busses ist prioritätsgesteuert - entweder nach dem Prinzip der Kettenpriorität (Daisy-chain-Priority) oder in paralleler Form. Dies erfordert eine Bus-Arbitration-Logik bei jedem Master, um zu verhindern, daß die Kontrolle über den Bus mehr als einem Master zur gleichen Zeit zugeordnet wird. Ein Bus-Takt bestimmt den Zeitpunkt, zu dem die Buszuteilung bei mehrfachen Anforderungen vorgenommen wird. Nach der Buszuteilung wird nach dem oben erwähnten Quittungsverfahren der Datentransfer nur noch von den beteiligten Systemmodulen bestimmt. Dabei können Einfach- oder Mehrfachübertragungen vorgenommen werden, sobald ein Master die Kontrolle über den Bus erlangt hat.

Mit der Leistungsfähigkeit und Flexibilität des MULTIBUS-Systembusses ist die Grundlage für ein breites Spektrum von Anwendungsbereichen gegeben. Zum Beginn des Entwicklungsvorhabens war die Verfügbarkeit von Peripheriesteuergeräten (Massenspeicher) mit entsprechender Softwareunterstützung und eines Echtzeit-Multitasking Executives (RMX80, vgl. Kap. 10.1) ein schwerwiegendes Entscheidungskriterium, da kein anderes Multiprozessorsystem zu diesem Zeitpunkt diesen für den vorgesehenen Einsatz so wichtigen Systembaustein zur Verfügung stellte.

Für den Aufbau von Automatisierungssystemen in der europäischen Industrie hat sich bezüglich der mechanischen Abmessungen das Europakartenformat durchgesetzt. Dieses wird jedoch von einem konventionellen MULTIBUS-System nicht befolgt. Das führte u.a. zu einer nur langsamen Einführung von INTEL iSBC-Systemen in die deutsche Industrie. Da das

Entwicklungsvorhaben neben dem Einsatz in der Pilotanlage auch zu einer breiteren Vermarktung vorgesehen war, wurde deshalb nach einem System gesucht, daß auch den geforderten mechanischen Standards genügt.

7.2 Multicomputerfähiges Baugruppensystem AMS

Die Firma Siemens hat im Jahre 1979 den Aufbau eines Baugruppensystems für Multicomputeranwendungen veröffentlicht /7/8/, das um den mehrprozessorfähigen AMS-BUS aufgebaut ist. Dieser vereint die Flexibilität und Leistungsfähigkeit des MULTIBUS-Systembusses mit der Forderung nach Ausführung in Eurocard-Standard. Der AMS-BUS ist funktionell aufwärts-

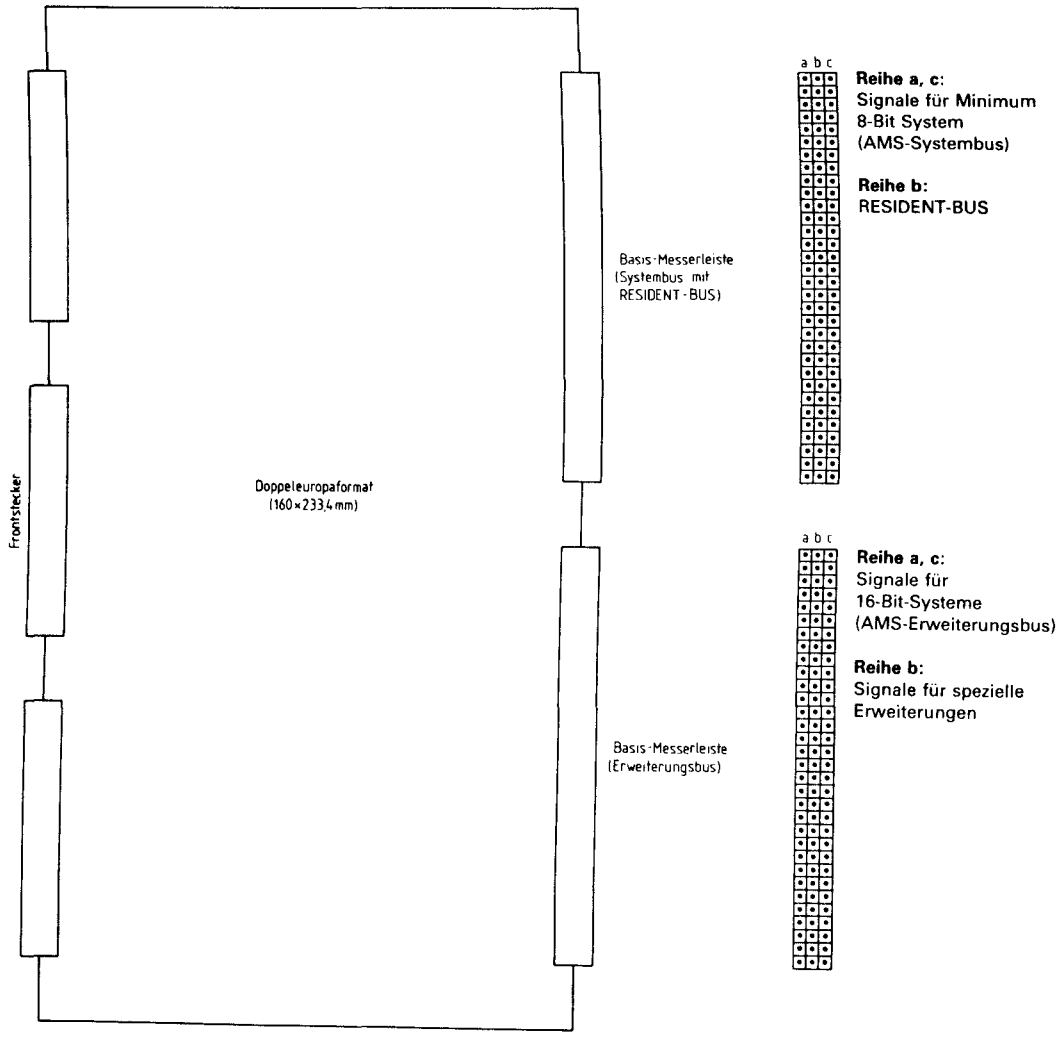


Abb. 7.2-1 Mechanischer Aufbau der AMS-Baugruppen

kompatibel zum Intel MULTIBUS und damit in allen hauptsächlichen Funktionen und Zeitbedingungen kompatibel zum IEEE 796 Bus.

Der AMS-BUS verbindet durch eine mehrlagige Bus-Leiterplatte in zweizeiligen 19"-Baugruppenträgern (Doppeleuropaformat) die einzelnen Systemmodule. Die Verbindung der Baugruppen geschieht nicht wie beim MULTIBUS-Konzept über direkte, sondern indirekte Steckung mittels zweier VG-Stecker (DIN 41612). Die Signalleitungen verteilen sich auf diese beiden Stecker in Form eines System-Busses und eines Erweiterungsbusses (vgl. Abb. 7.2-1). Der Systembus enthält dabei alle Adreß-, Daten- und Signalleitungen für einen Standard 8 Bit Einplatinenrechner mit 64 KByte Adreßraum. Für einen erweiterten Adreßraum bis zu 8 Megabyte und 16 Bit Mikroprozessoren steht der Erweiterungsbus zur Verfügung. Zusätzlich ist ein Multiplexbus (RESIDENT-BUS) vorgesehen, der auf dem Stecker des Systembusses untergebracht ist. Er dient hauptsächlich der lokalen Erweiterung von Systemmodulen und ist deshalb auch nicht multicomputerfähig (weitere Informationen gibt /9/). Die beiden Bussysteme AMS-BUS und RESIDENT-BUS bilden zusammen das AMS-BUS-System.

7.3 Verteilte MULTIBUS-AMS-Architektur

Durch die funktionelle Kompatibilität des AMS-BUS zum MULTIBUS kann ein AMS-System leicht durch das große Angebot von MULTIBUS-Platinen ergänzt werden. Um beide Systeme ineinander überzuführen, ist lediglich ein passiver Adapter erforderlich. Bei der Auslegung der AMS-Master-Baugruppen (8 Bit Prozessoren) wurde auf weitgehende Systemverträglichkeit zum RMX80 Multitasking-Executive besonderer Wert gelegt (ein ähnliches System wird von Siemens unter dem Namen RTOS vertrieben /10/). Deshalb ist die Entwicklung eines einheitlichen Betriebssystems auf Grundlage des RMX80 möglich, das in einem verteilten System auf verschiedenen Mikrorechnerclustern ablauffähig ist, die jeweils für sich

auf der Basis eines AMS-BUS oder MULTIBUS aufgebaut sein können. Jeder Knoten des verteilten Systems kann dann mit dem Multiprozessor-System ausgerüstet werden, das den jeweiligen Anforderungen am weitesten entgegenkommt.

Wegen der in einem MULTIBUS-System 1978 bereits verfügbaren Peripheriegerätesteuerungen (Disk-Controller) wurde deshalb der Supervisor ausschließlich mit käuflichen Intel MULTIBUS-Komponenten ausgestattet, während die einzelnen Prüfstandssteuereinheiten auf der Grundlage des AMS-Systems

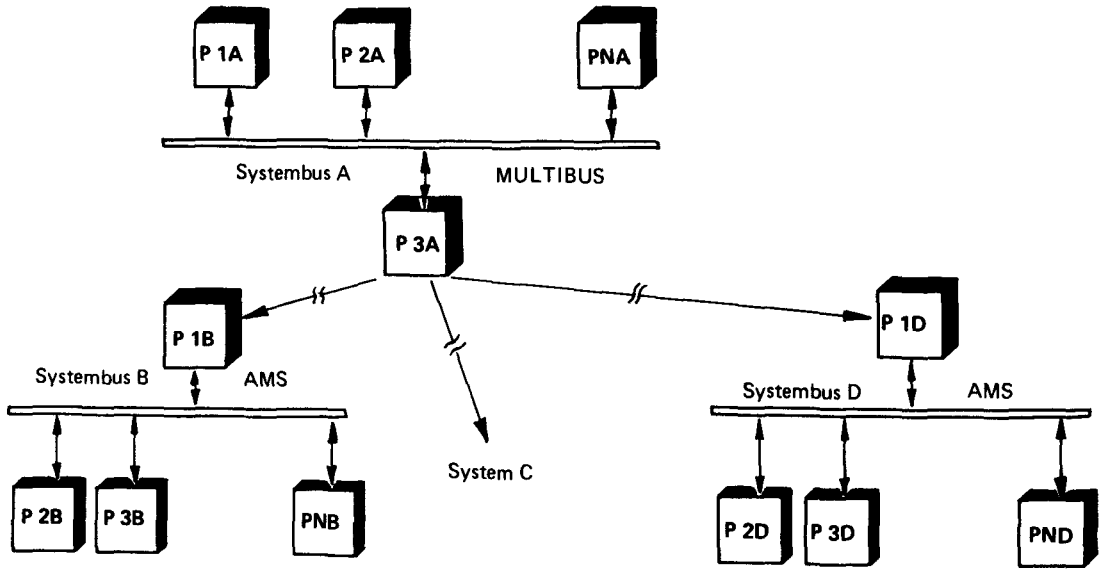


Abb. 7.3-1 Multiprozessor-Systemstruktur mit hierarchisch organisierter Prozeßleitfunktion und drei dezentralen Prüfstandssteuereinheiten

entwickelt wurden. Diese bestehen im wesentlichen aus käuflichen Baugruppen (vgl. Kap. 8.2), die durch wenige Hardwareeigenentwicklungen ergänzt wurden. Eine der käuflichen AMS-Baugruppen ist mit einem Arithmetik-Prozessor (AMD 9511) ausgerüstet. Dies hat die Entscheidung für den Einsatz des AMS-Systems in den Prüfstandssteuereinheiten (neben der Europakartenausführung) ebenfalls beeinflußt, da für die digitale Regelung (DDC) eine schnelle Arithmetik Voraussetzung ist.

8. HARDWARE KONFIGURATION

8.1 Zentrales Rechnersystem

Die Konfiguration des Zentralrechnersystems ist in Abb. 8.1-1 dargestellt. Es besteht aus einem Intel iSBC Mikrorechnersystem mit Floppy-Disk-Massenspeicher, einer Bedienerkonsole (HP2647A /11/) mit angeschlossenem HP2631-Drucker /12/ zur Prüfprotokollerstellung und einem Betriebsprotokolldrucker LA34 der Firma Digital Equipment. Alle diese Komponenten waren auf dem Markt erhältlich und konnten ohne jede weitere Hardwareentwicklung eingesetzt werden. Im folgenden werden die wesentlichen Eigenschaften der Multibus-Systemkomponenten erläutert, eine genauere Beschreibung findet sich in der sehr umfangreichen Dokumentation der Firma Intel.

Nach Abb. 8.1-1 wird das Mikrorechnersystem von einer Master-Baugruppe (iSBC 80/30) gesteuert, während zwei mit eigenen µ-Prozessoren bestückte Slave-Funktionseinheiten die komplexen Aufgaben der Floppysteuerung und Kommunikation zu den drei unterlagerten Prüfstandssteuereinheiten wahrnehmen. Das System wird durch zwei passive Baugruppen, einem zentralen "Mailbox-Speicher" und einem Kommunikationscontroller zum Anschluß des Betriebsprotokolldruckers ergänzt. Die Kommunikation der intelligenten µ-prozessorbestückten Baugruppen wird durch die auf den Platinen integrierten "Dual Port" Speicher und die "Mailbox" unterstützt. Dabei geschieht der Datenaustausch im programmierten Transfer. Eine Ausnahme bildet das Floppy-Steuergerät, das die Nettodaten über direkten Speicherzugriff (DMA) austauscht.

Der iSBC 80/30 Einkarten-Mikrorechner repräsentiert die oberste gesamtsystembezogene Verwaltungsebene des Automatisierungssystems. Er ist auf Basis des 8085A µ-Prozessors mit 8 Bit Wortbreite aufgebaut und verfügt über einen lokalen Speicher von 8 KByte (EPROM) und einen "Dual-Port" Speicher mit 16 KBytes. Zum Anschluß der Bedienerkonsole

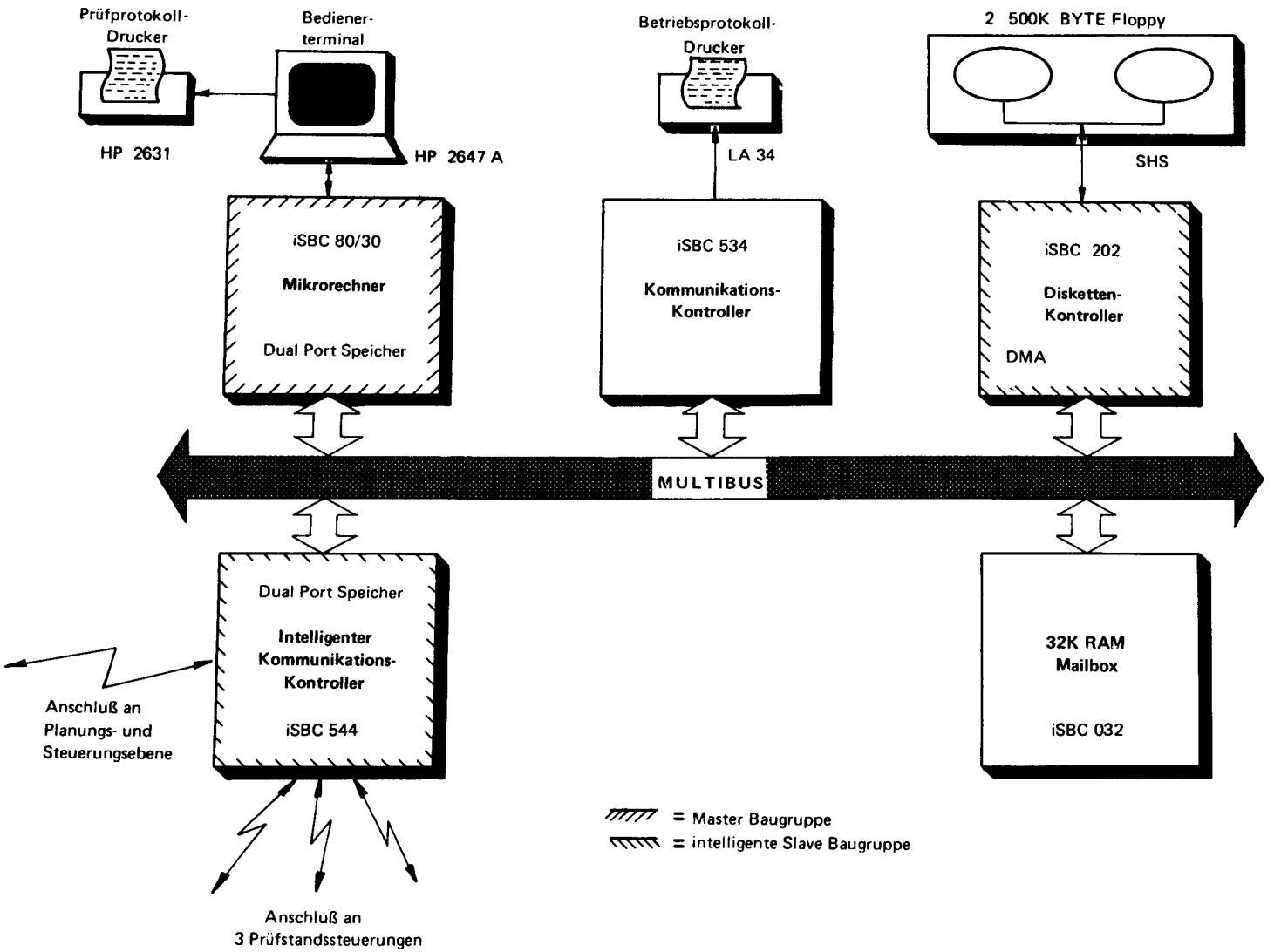


Abb. 8.1-1 Konfiguration Zentralrechnersystem

ist ein serieller Kanal mit programmierbarer Übertragungsgeschwindigkeit integriert. Zwei 16 Bit binär oder BCD Zähler unterstützen in der Anwendung das RMX80 "Multi-Tasking-Executive" (vgl. Kap. 10). Die integrierte MULTI-BUS-Kontrollogik erlaubt den Zugriff des Mikrorechners auf den Systembus als aktiver Teilnehmer.

Der interne Speicher des Mikrorechners wird durch die verwendete Speicherbaugruppe iSBC 032 um 32 KByte erweitert. Es handelt sich dabei um einen dynamischen Speicher mit integrierter "Refresh" Logik. Die Start-Adresse des Speicherbereiches läßt sich in Segmenten von 16 KByte über Vorverdrahtung wählen.

Da der eingesetzte Mikrorechner (iSBC 80/30) nur einen seriellen Ausgabekanal besitzt, wird zum Anschluß des Betriebsprotokolldruckers eine zusätzliche Baugruppe (iSBC 534 Communication Controller) eingesetzt, der als passiver Teilnehmer die serielle Ausgabemöglichkeit des Mikrorechners um weitere 4 Kanäle erweitert.

Zum Anschluß der Prüfstandssteuereinheiten und eines übergeordneten Planungs- und Steuerungssystems muß eine gesicherte Kommunikationsprozedur zur Verfügung gestellt werden. Da diese den zentralen Mikrorechner zeitlich zu sehr belasten würde, wurde ein intelligenter Kommunikationskontroller eingesetzt (iSBC544), der den Anschluß von vier seriellen Kanälen gestattet. Die Steuerung der gesicherten Prozedur /13/ und die Pufferorganisation werden autonom behandelt mit Hilfe eines 8085A Mikroprozessors. Ein 16 KByte "Dual Port" Speicher dient im Wesentlichen dem Austausch von Datenblöcken zwischen den Kommunikationsstrecken (bzw. Prüfstandssteuereinheiten) und dem zentralen Mikrorechner. Auch hier läßt sich der Speicher wieder in 16 KByte Segmenten der Adressierung des MULTIBUS-Systembusses zugänglich machen. Drei unabhängige programmierbare Timer/Zähler dienen dem RMX80 Betriebssystemkern und der Überwachung der Kommunikationskanäle.

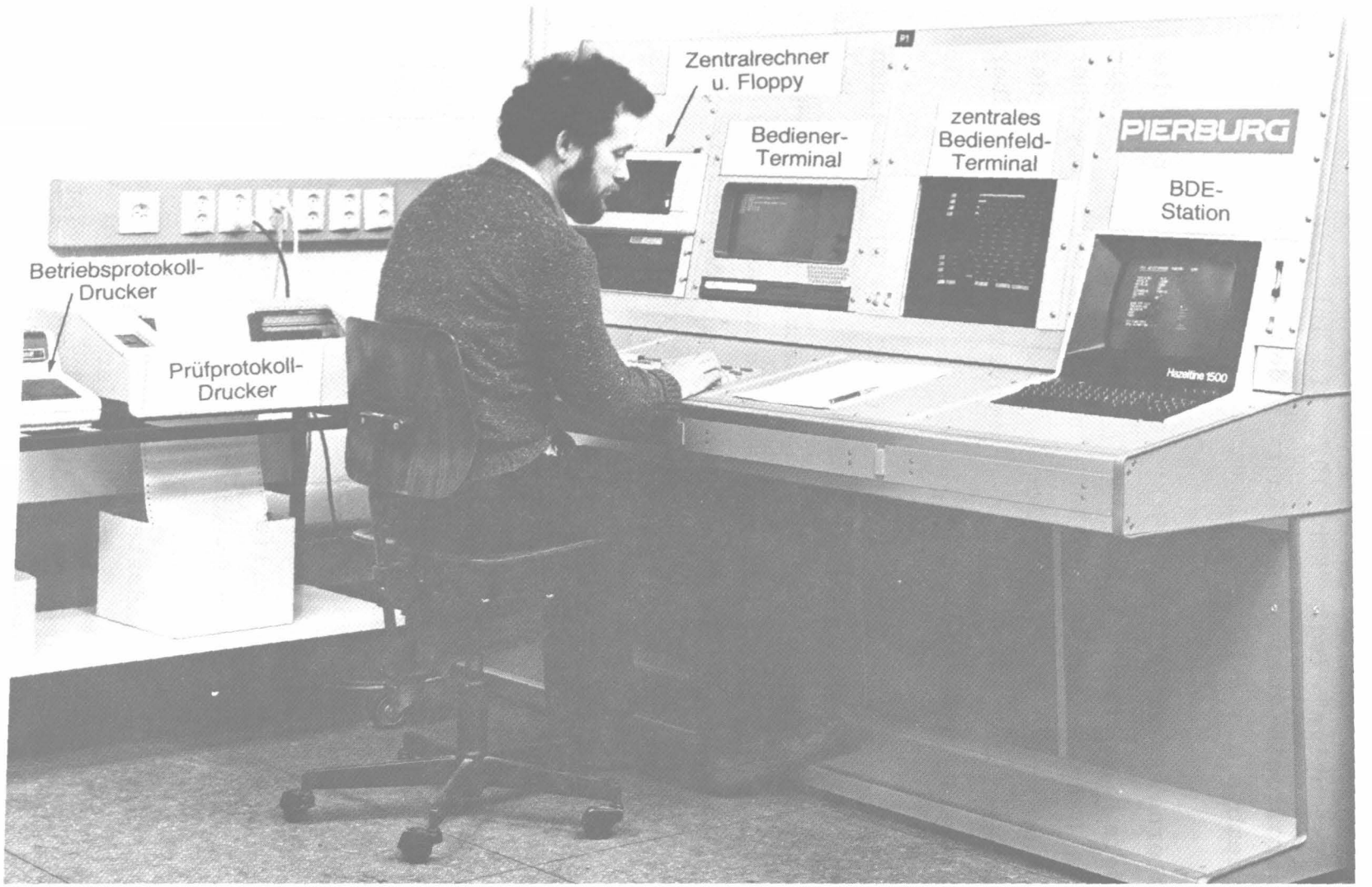


Abb. 8.1-2 Zentralrechnersystem mit BDE-Station und zentralem Bedienfeldterminal

Die eingesetzten zwei Floppy-Laufwerke mit doppelter Speicherdichte (jeweils 500 KByte) werden über die Peripheriesteuerung (iSBC202 Double Density Disk Controller) mit dem Systembus verbunden. Es handelt sich dabei um eine für sich abgeschlossene Funktionseinheit, die mit Bitslice Prozessoren der Intel-Serie 3000 aufgebaut ist. Eine DMA-Steuerung erlaubt den Blocktransfer von Daten von und zum Massenspeicher parallel zur Arbeit des iSBC 80/30 Mikrorechners. Eine leistungsfähige Softwareunterstützung für das auf zwei getrennten Baugruppen realisierte Peripheriesteuergerät wird von den Systemdiensten des RMX80 (vgl. Kap. 10) bereitgestellt.

In Abb. 8.1-2 ist das Zentralrechnersystem mit zugehöriger Peripherie dargestellt. Zusätzlich wurde in das pultförmige Gehäuse ein Bedienfeldterminal und eine Betriebsdatenerfassungsstation mit Ausweisleser integriert. Diese ist an das produktionsüberwachende Steuerungssystem angeschlossen, das in /14/15/ näher beschrieben ist (vgl. auch Abb. 13.1). Zu einem späteren Zeitpunkt sollen die auftragsbezogenen Daten, die über die BDE-Station gewonnen werden, auch mit den gewonnenen Meßdatenfiles verkettet werden. Auf dem zentralen Bedienfeldterminal (Farbmonitor) läßt sich jeder Prüfstand anwählen und dessen aktuelles Monitorbild darstellen (Verbindung über Koax-Leitungen).

8.2 Prüfstandssteuereinheit

Die Prüfstandssteuereinheit läßt sich in drei voneinander auch im mechanischen Aufbau trennbare Einzelbereiche gliedern:

- Das AMS-Multiprozessorsystem mit der zugehörigen Bedienerschnittstelle

- Die Anpaßelektronik mit galvanischer Entkopplung, Vorverstärkern, Relais, Treibern für Strom, Spannungen und Ex-Barrieren, über die nach TÜV-Vorschriften alle

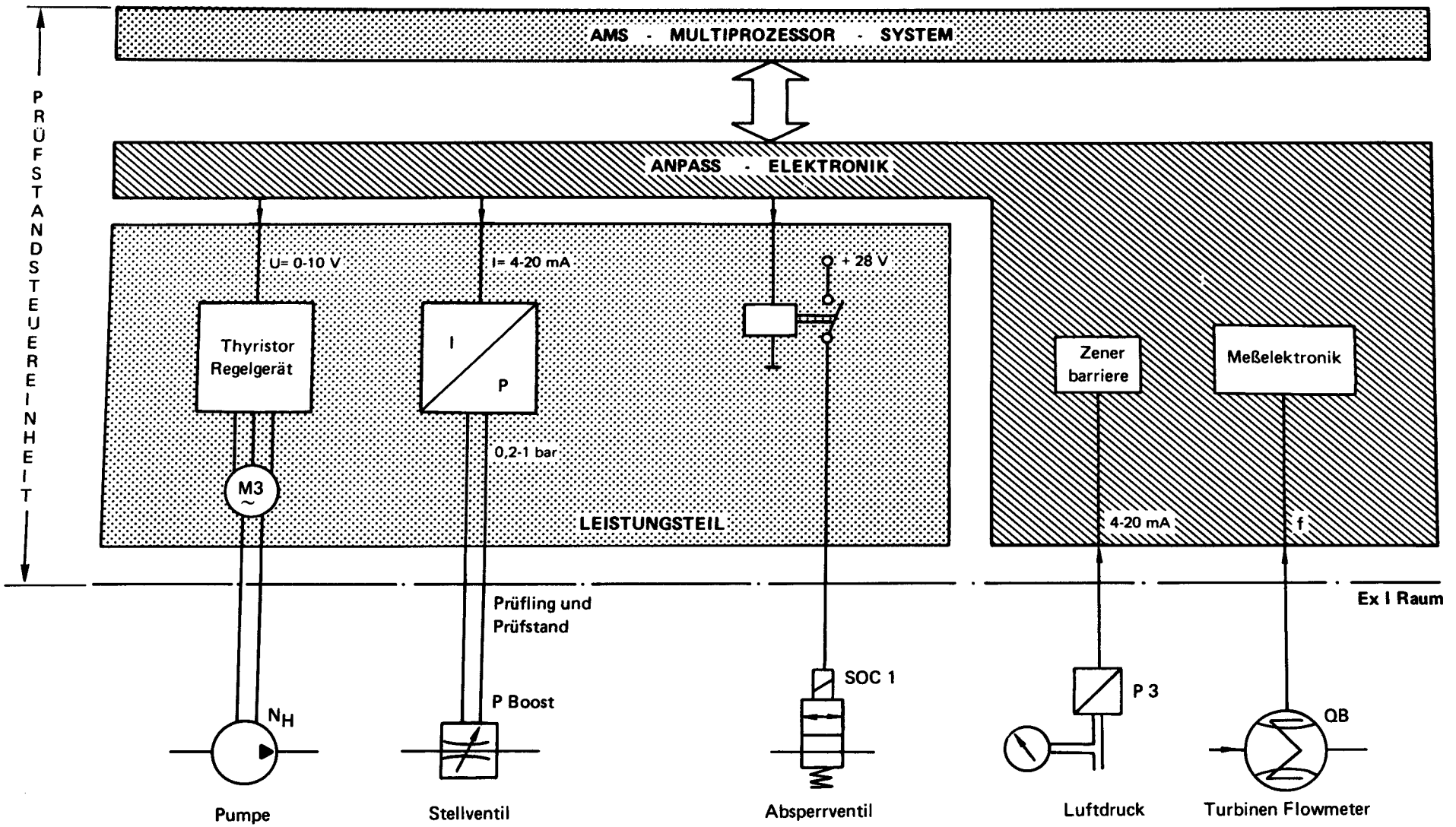


Abb. 8.2-1 Prüfstandsteuereinheit

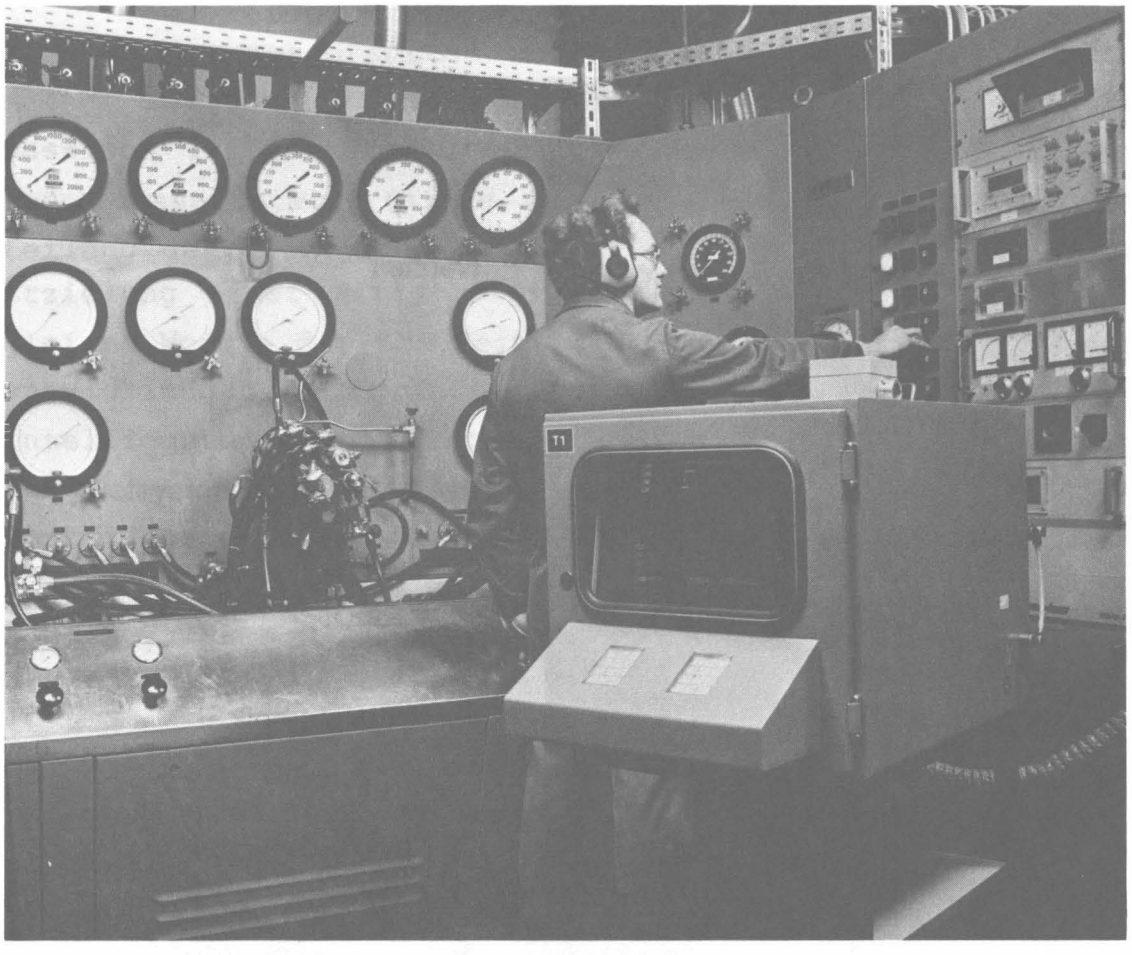


Abb. 8.2-2 Prüfstand mit ex-geschütztem Bedienfeld

elektrischen Signale in bzw. aus dem explosionsgefährdeten Bereichen zu führen sind.

- Der Leistungsteil mit Strom/Druck-Umsetzern zum Ansteuern von Ventilen und unterlagertem Thyristorregler mit Phasenanschnittsteuerung zur Anschaltung des Hauptantriebsmotors.

Die Betriebsart "Manuell" wird durch Umschaltung am Leistungsteil vorgenommen, so daß die Elektronikkomponenten des Multiprozessorsystems vollständig von dem Prüfstand abgetrennt werden können.

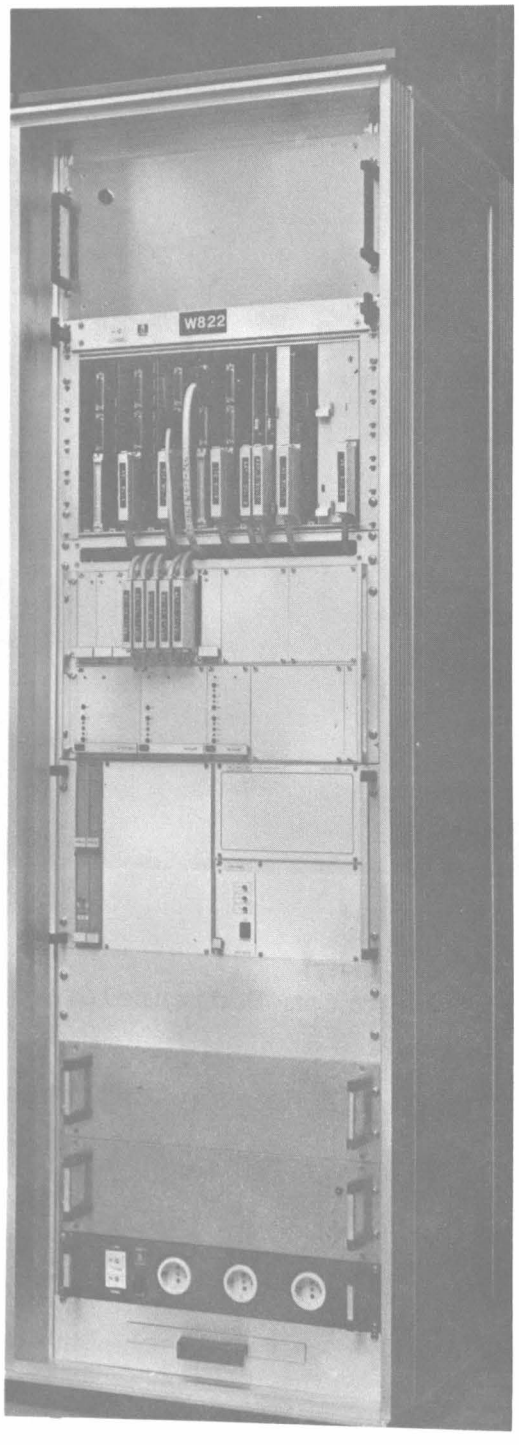


Abb. 8.2-3 Prüfstandsteuereinheit ohne Leistungsteil

8.2.1 Multiprozessorkonfiguration

Das Multiprozessorsystem der Prüfstandsteuereinheit wurde mit 5 Masterbaugruppen, einem intelligenten Slave, drei passiven Teilnehmern und einer Baugruppe zur Systemsteuerung ausgerüstet. (siehe Abb. 8.2.1-1). Die Aufgabenverteilung geschieht funktionsorientiert (vgl. Kap. 6.7). Zur Erzielung einer wirtschaftlichen Lösung wurden weitgehend gleiche Module der AMS-Reihe verwendet, die Hardware-Eigenentwicklungen beschränkten sich auf die Analog Ausgabe, Intelligente Analog Eingabe und Steuerungskarte für "Power Fail" und Busprioritätslogik. Die anwendungsorientierten Funktionseinheiten wurden weitgehend per Software implementiert und umfassen zum Teil mehrere Baugruppen (siehe Abb. 8.2.1-1 DDC-Regler).

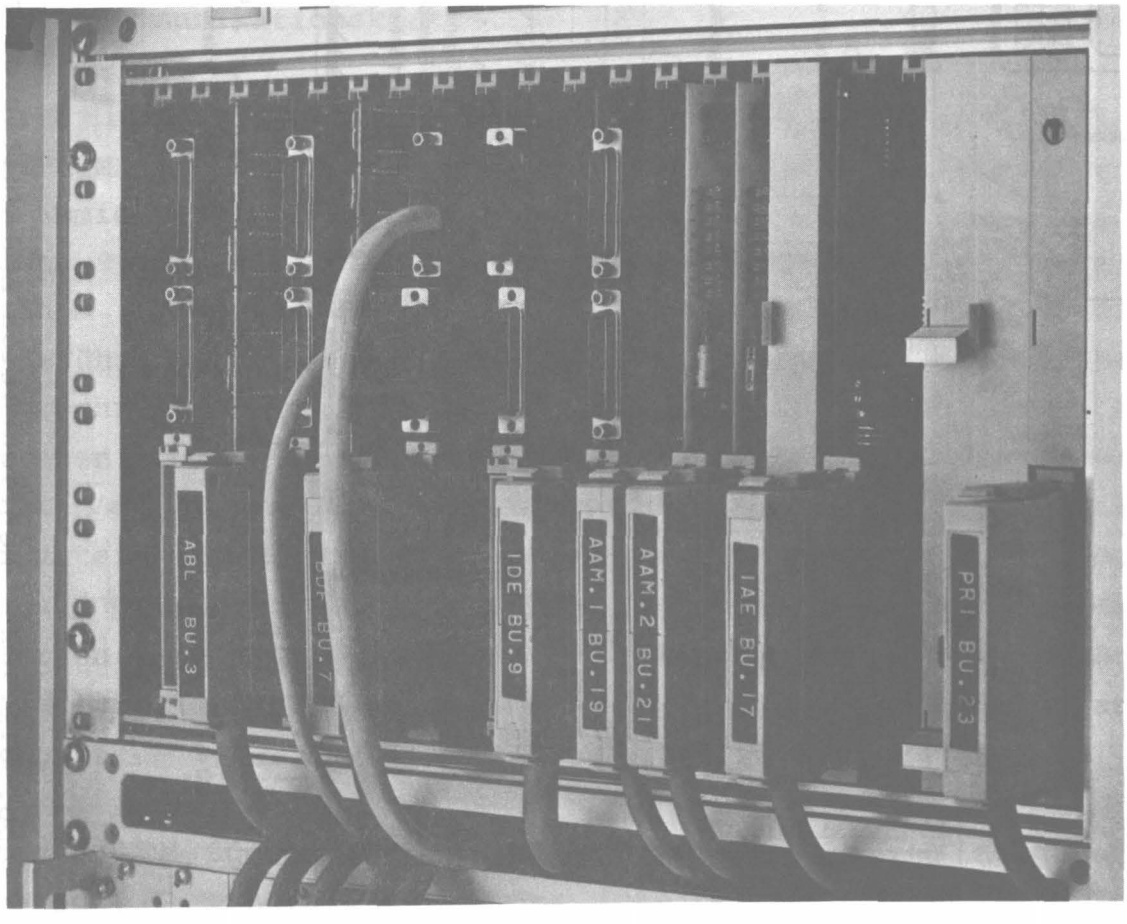


Abb. 8.2.1-1 Aufbau AMS-Multiprozessorsystem

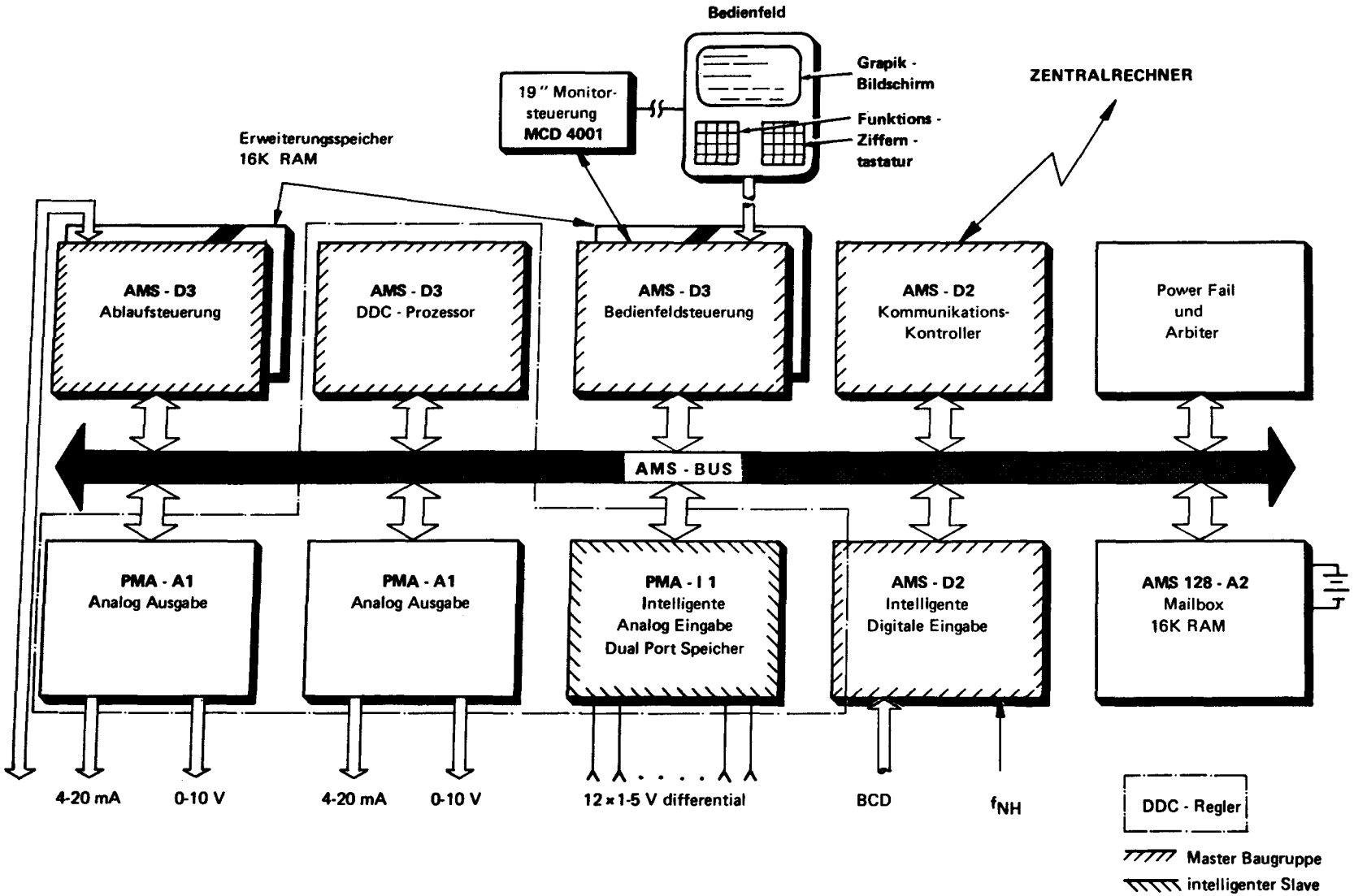


Abb. 8.2.1-2 Konfiguration AMS-Multiprozessorsteuereinheit

Trotz der durch Anwendung der Mehrlagentechnik bedingten hohen Packungsdichte war die Integration von "Dual Port" Speichern auf den Platinen der AMS-Mikrorechner nicht möglich. Deshalb wurde eine zentrale "Mailbox" zur Interprozessor Kommunikation eingesetzt (vgl. Kap. 9.3).

8.2.1.1 Universeller Mikrorechner AMS-D2/D3, Basis für die wichtigsten Funktionseinheiten

Der Einplatinenrechner AMS-D2/D3 dient als standardisierte Baugruppe für folgende Funktionseinheiten:

- Ablaufsteuerung (ABL)
- Bedienfeldsteuerung (BDF)
- Intelligente digitale Eingabe (IDE)
- DDC Prozessor (DDC)
- Kommunikationskontroller (COM)

Die Mikrorechner enthalten neben Zentraleinheit, Speicher (EPROM und RAM), Ein/Ausgabe (parallel und seriell), programmierbare Zeitgeber und Interrupt-Steuerung (vgl. Abb. 8.2.1.1-1). Die Implementationsvariante AMS-D3 ist mit einem Arithmetik-Prozessor (AM9511A) ausgerüstet, der die Ausführung elementarer und komplexerer mathematischer Berechnungen im Fest- und Gleitpunkt-Format erlaubt. Es stehen die Grundrechenarten und mathematische Funktionen zur Verfügung. Als Koprozessor arbeitet der Arithmetik-Baustein parallel zum eingesetzten µ-Prozessor 8085A. Bei baugruppeninternem Datenverkehr und Datenaustausch mit externen Baugruppen wird die Geschwindigkeit durch die Ansteuerung des READY-Eingangssignals des µ-Prozessors angepaßt.

Der EPROM Speicherbereich kann bis zu 12 KByte ausgebaut werden. Der Adreßbereich des lokalen RAM umfaßt 4 KByte. Dieser Speicherausbau erwies sich als zu gering für die Implementation der zum Teil sehr umfangreichen anwendungsorientierten Funktionen. Deshalb wurde eine Zusatzkarte

entwickelt, die den Speicherbereich auf 32 KByte EPROM und bis zu 16 KByte RAM erweitert. Dazu wurde der lokale Bus der Baugruppe AMS-D2/D3 auf eine Sandwich Platine geführt, die mit den erforderlichen Speicherbausteinen und einem hochintegrierten Refresh-Controller bestückt ist (siehe Abb. 8.2.1.1-1).

Die parallele Ein/Ausgabe gestattet 48 periphere Anschlüsse, die mit Treiberbausteinen und Widerstands-Moduln ausrüstbar sind.

Die serielle Ein/Ausgabe liefert wahlweise die Signale einer V24/V28 oder einer 20 mA Schnittstelle mit programmierbarer oder durch eine externe Taktquelle bestimmbarer Übertragungsrate. Die 20 mA Schnittstelle bietet im Sender eine Stromquelle und ist im Empfänger über einen Optokoppler galvanisch getrennt.

Zwei Zeitgeber stehen dem Anwendungsprogrammierer zur Verfügung. Die Ausgänge führen in die Interruptsteuerung und können zur Programmunterbrechung benutzt werden. Die vier Interrupt-Eingänge des eingesetzten Mikroprozessors werden durch acht weitere der Interruptsteuerung ergänzt.

Die parallelen Ein/Ausgabe-Kanäle werden in der Ablaufsteuerung (ABL) zur Bedienung der binären Prüfstandsfunktion verwendet (vgl. Abb. 8.2.1.1-1). Die vom Prozeß gelieferten BCD-Meßwerte werden in der intelligenten Digitalen Eingabe (IDE) über die parallele Schnittstelle des Mikrorechners angeschaltet. Das frequenzproportionale Drehzahlsignal (fNH) wird in derselben Funktionseinheit in einem der Zähler/Zeitgeber integriert. Die Bedienfeldsteuerung (BDF) ist mit der Funktions- und Zifferntastatur des Bedienfeldes ebenfalls über die periphere parallele Schnittstelle verbunden. Die serielle V24 Schnittstelle wird zur Kopplung der Monitorsteuerung MCD4001/16/ verwendet. Der Kommunikationscontroller wird über die 20 mA Linienstromschnittstelle mit dem zentralen Rechnersystem verbunden. Die Aufgaben des DDC-Prozessors werden wesent-

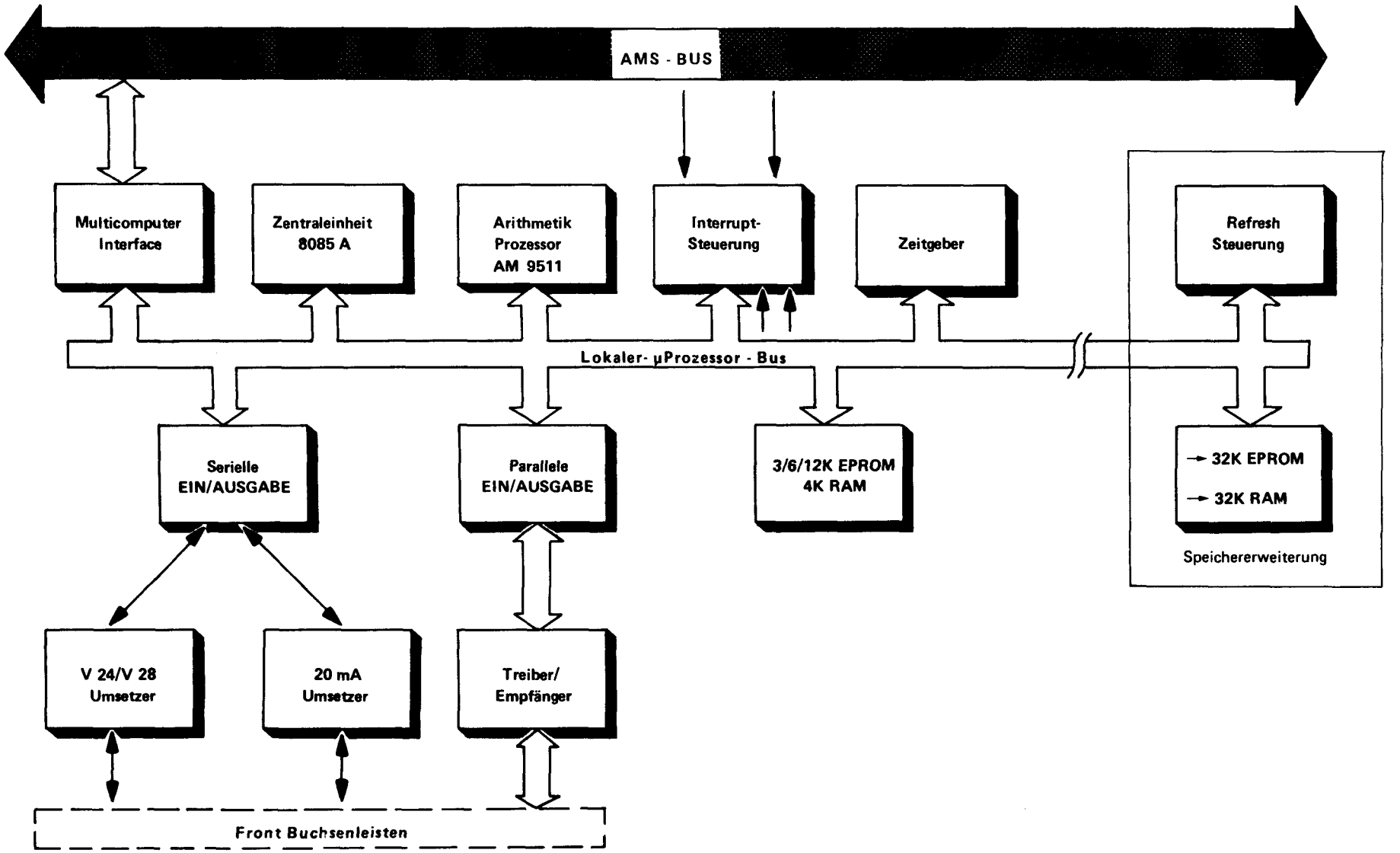


Abb. 8.2.1.1-1 Blockschaltbild AMS-D3 Mikrorechner mit Speichererweiterung

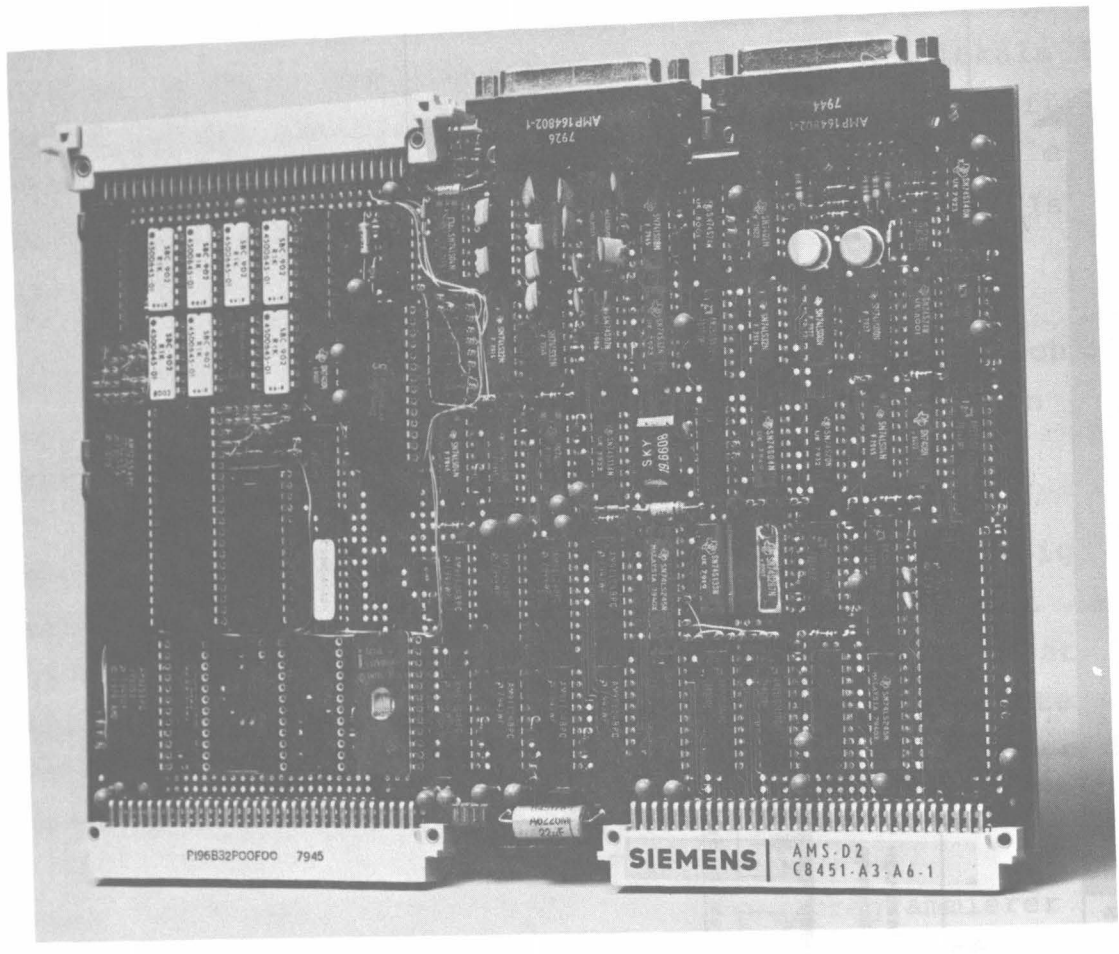


Abb. 8.2.1.1-2 Mikrorechnerkarte AMS-D2

lich von dem Arithmetikprozessor des AMS-D3 Mikrorechners unterstützt.

8.2.1.2 Intelligente Analog-Eingabe (IAE)

Das µ-Prozessor gesteuerte Analog-Eingabe-Modul dient der Erfassung, Vorverarbeitung und systemkonformen Bereitstellung von Prozeßmeßwerten, die als elektrische Signale vom Prüfstand geliefert werden. Im Rahmen des Multiprozessorsystems stellt das Modul einen intelligenten "Slave" dar, auf den die DDC-Master-Baugruppe über das integrierte "Dual Port RAM" zugreifen kann (vgl. Abb. 8.2.1.2-1). Die Zentraleinheit (8085A) ist mit dem Arithmetik Koprozessor (AM9511) und einem hochkomplexen Datenerfassungssystem (Burr Brown SDM857) über den lokalen Bus verbunden.

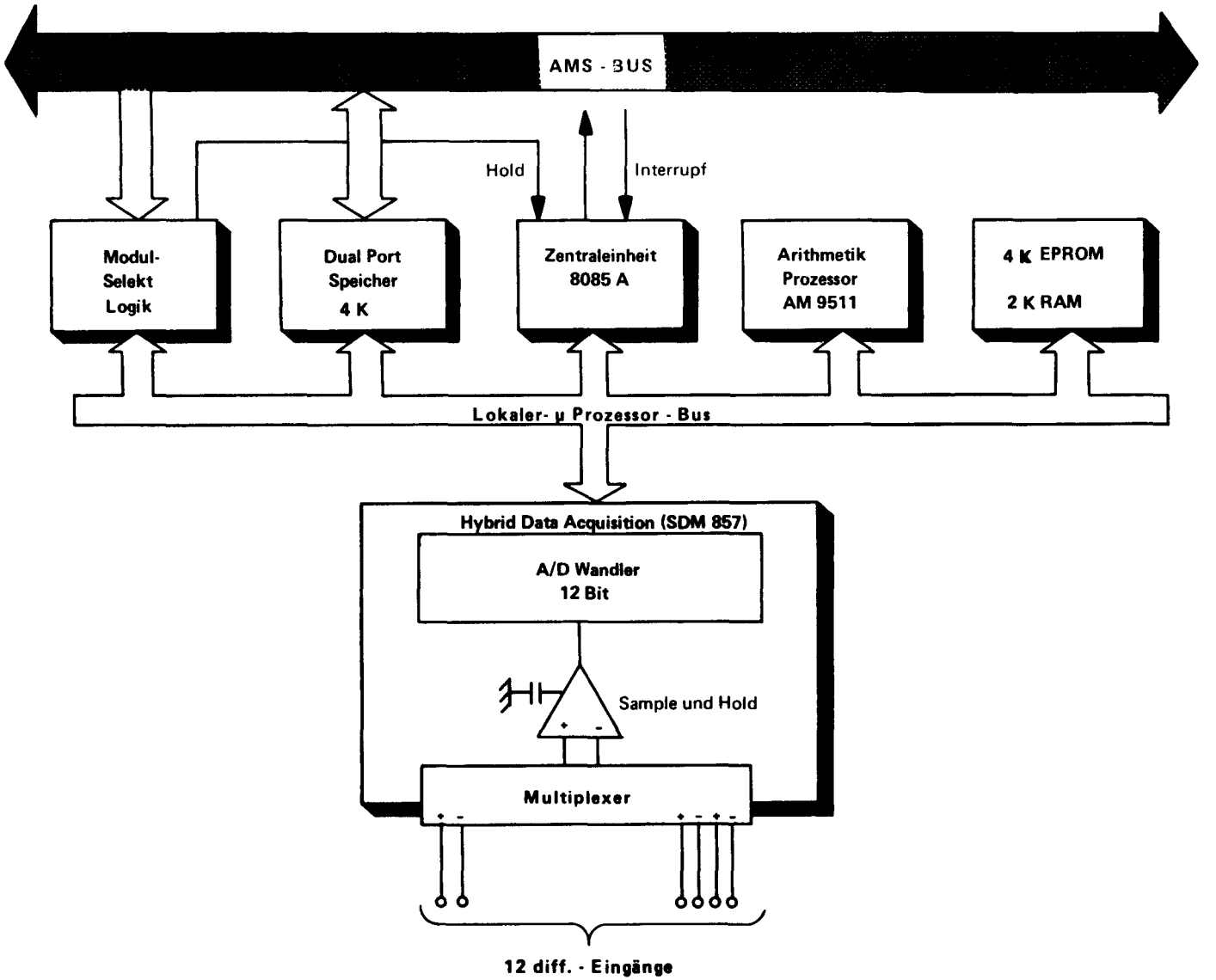


Abb. 8.2.1.2-1 µ-Prozessor-gesteuerte Analog-Eingabe

Das "Data Acquisition System" enthält alle erforderlichen Komponenten, um analoge Signale in ein äquivalentes digitales Ergebnis umzuwandeln. Es stehen 30 massebezogene Eingänge oder 15 Differenzeingänge zur Verfügung. Der Spannungsbereich ist wählbar zwischen 0 - 10 V, $\pm$ 5 V und $\pm$ 10 V. Die max. zulässige Eingangsspannung beträgt $\pm$ 20 V. Die Genauigkeit ist wählbar zwischen 8, 10 und 12 Bit. Die dabei zu erreichende Umsetzrate liegt zwischen 29 kHz (12 Bit) und 70 kHz (8 Bit). Weiterhin ist die Verstärkung des internen Instrumentenverstärkers über Widerstände im Bereich zwischen 2 und 500 wählbar. Diese unterschiedlichen Betriebsarten werden mittels Wrap-Brücken eingestellt. Mit dem lokalen Prozessorbus ist das System über Tri-State-Ausgänge verbunden.

Mit dem Arithmetikprozessor AM 9511 (APU) ist das IAE-Modul in der Lage, mathematische Berechnungen im Fest- und Gleitkommaformat durchzuführen, wie z.B. Skalierung, Linearisierung, gleitende Mittelwertbildung usw.

Der Datenaustausch mit dem, externen Master erfolgt über die Modul-Select-Logik und das erwähnte Dual-Port RAM von 4K Byte. Die Startadresse des Dual-Port RAMs wird über Wrap-Brücken festgelegt. Bei einem Zugriff des externen Masters auf das Dual-Port RAM generiert die Modul-Select-Logik ein Hold-Signal für den Mikroprozessor. Als Folge dieses Hold-Signals werden die internen Daten- und Adreßtreiber in den Tri-State Zustand geschaltet und die entsprechenden Treiber zum AMS-Bus aktiviert. Ist der Masterzugriff beendet, werden Daten- und Adreßtreiber umgeschaltet und das Hold-Signal zurückgenommen.

Über eine Bus-Interrupt-Leitung kann der externe Master dem IAE-Modul eine Datenanforderung mitteilen. Es quittiert diese Anforderung mit einem "Daten-gültig" Signal ebenfalls über eine Bus-Interrupt-Leitung.

8.2.1.3 Analog-Ausgabe (AA)

Das Analog-Ausgabemodul stellt einen passiven "Slave" im Multiprozessorsystem dar und ermöglicht die optoentkoppelte Ausgabe von 4 analogen Spannungen bzw. Strömen mit einer Auflösung von 12 Bit. Die Anpassung an den Multiprozessor-Bus erfolgt über ein Bus-Interface. Der gewünschte Ausgabewert, der als 12 Bit Integer vorliegt, wird für jeden Kanal in einem Register von 12 Bit Breite zwischengespeichert. Vor den Registern wird über Optokoppler eine Potentialtrennung vorgenommen. Die Registerausgänge sind mit den Eingängen eines Digital-Analogwandlers verbunden, der den 12 Bit Wert in eine analoge Spannung umwandelt. Über einen Spannungs-Stromwandler wird die analoge Spannung (0 - 10 V) in einen proportionalen Strom (4 - 20 mA) umgeformt.

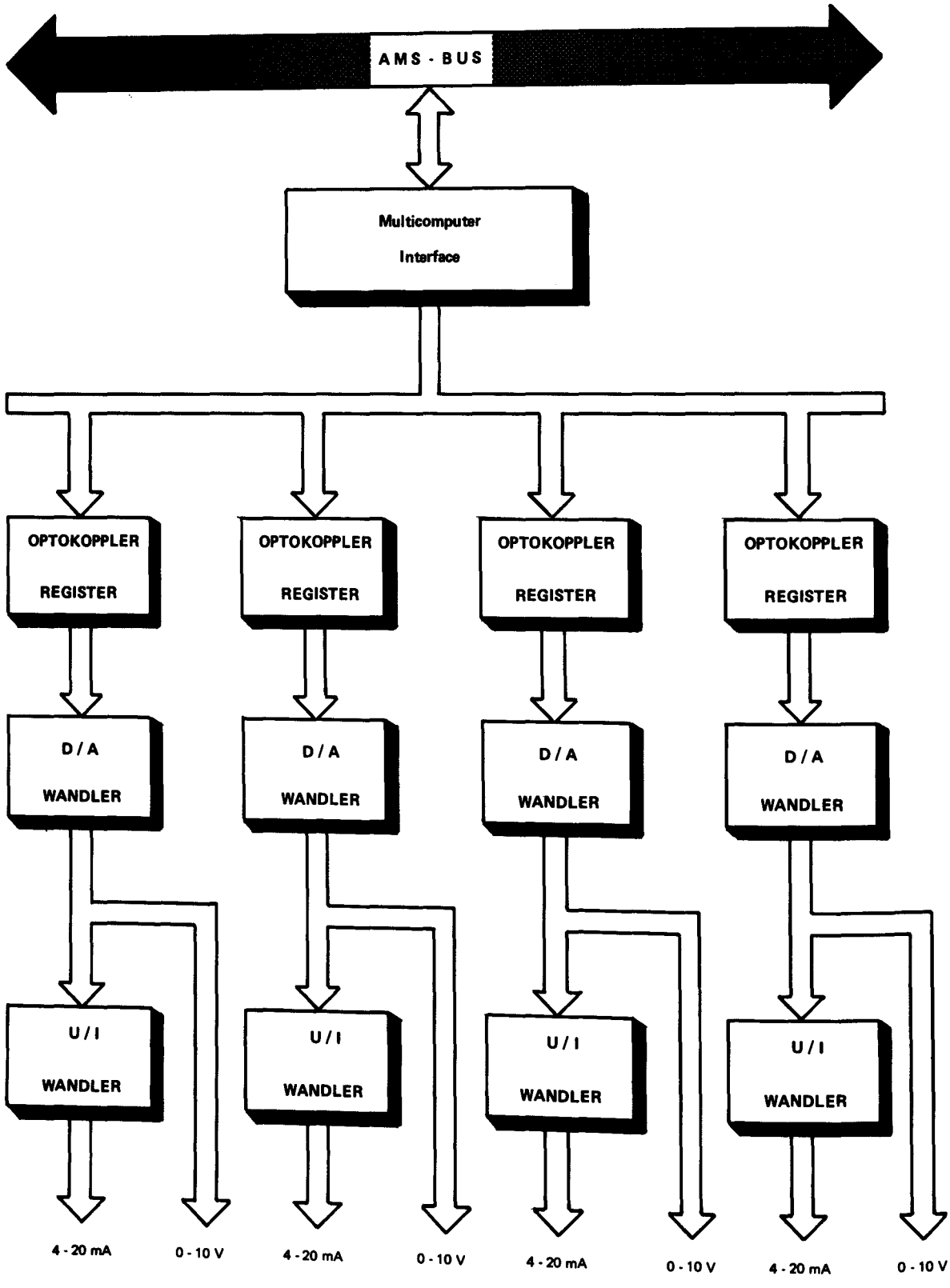


Abb. 8.2.1.3-1 Blockschaltbild Analog-Ausgabe

9. ÜBERSICHT PROGRAMMSYSTEM

In Abb. 9-1 ist ein ebenenorientiertes Modell des gesamten eingesetzten Programmsystems schematisch dargestellt. Dabei stellt jede Ebene der nächst höheren bestimmte Dienste zur Verfügung, benutzt dabei aber die Dienste der Ebenen darunter. Alle Details der unteren Ebene werden für die nächsthöheren unsichtbar. Dadurch erscheint jede Ebene mit den darunterliegenden als eine virtuelle Maschine mit nach oben hin wachsendem Abstraktionsgrad. Diese werden jeweils durch Schnittstellen realisiert, über die Aufrufe, zu den unteren Schichten gesendet und Antworten zurückgeschickt werden.

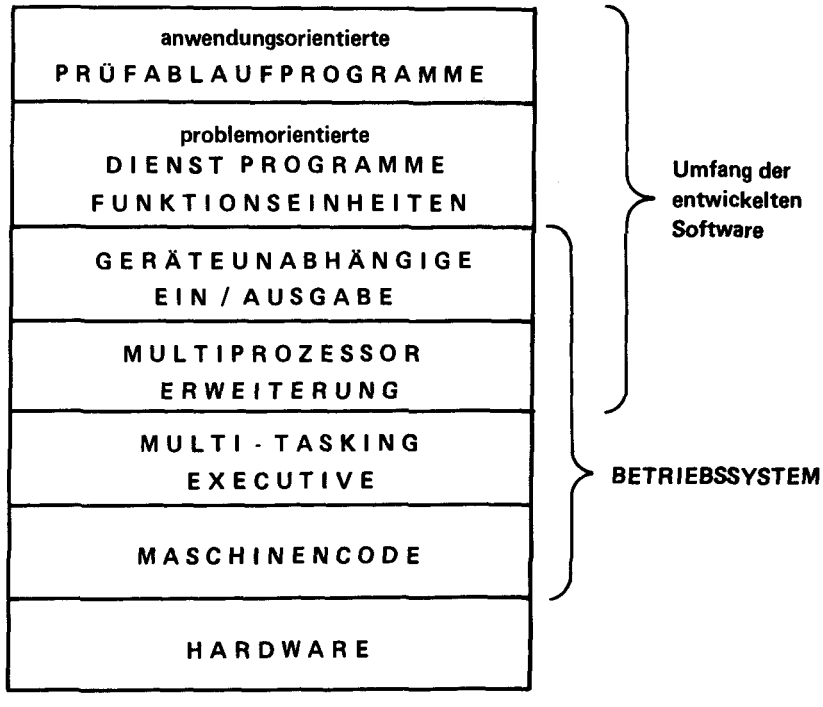


Abb. 9-1 Schichtenmodell des Programmsystems

Als höchste anwendungsorientierte Ebene ist das eigentliche Prüfablaufprogramm zu betrachten, das in der vorliegenden Implementierung mittels der Prüfablaufsprache "PRAS" (vgl. Kapitel 12) formuliert wird und dem Prüfingenieur eine besonders komfortable und flexible Anpassung an die jeweilige Prüfaufgabe erlaubt.

Die problemorientierten Dienstprogramme realisieren im wesentlichen auf Basis der verwendeten Hardware die in Abb. 6-1 dargestellten Funktionseinheiten, auf die das Prüfprogramm über transparente, leicht verständliche Aufrufe zugreifen kann.

Das Betriebssystem wurde um einen auf dem Markt verfügbaren Systemkern (Multi Tasking Executive, RMX80) mit Hilfe der erforderlichen Multiprozessorerweiterungen und einer Hochniveau-Schnittstelle ("Geräteunabhängige Ein/Ausgabe") implementiert, die auch die wesentlichen Kommunikationsfunktionen zwischen den drei Prüfstandssteuereinheiten und der Zentrale umfassen. Durch die "Geräteunabhängige Ein/Ausgabe" mittels logischer Kanäle wird ein botschaftsorientierter Kommunikationsmechanismus ermöglicht, der den funktionsorientierten Entwurf sehr effektiv unterstützt.

Bei der Implementierung der Betriebssystemfunktionen und der problemorientierten Dienstprogramme wurde weitgehend PLM /17/ verwendet, um zu einem methodisch strukturierten und modularen Programmsystem zu gelangen. Dabei wurde der erforderliche höhere Speicherbedarf (etwa Faktor 2 bis 4) und die damit verbundene langsamere Ausführungsgeschwindigkeit zugunsten der Vorteile in Kauf genommen, die eine höhere Programmiersprache während der Auslegung, Implementierung, Test, Dokumentation und nachträglichen Pflege bietet. Zur Sicherstellung einer genügend hohen Datenübertragungsgeschwindigkeit auf den seriellen Verbindungsleitungen zwischen den Prüfstandssteuereinheiten und dem Zentralrechnersystem wurden auf der untersten Kommunikationsebene (Serial Link Layer) die Programme in Assembler geschrieben /18/.

Die problemorientierten Dienstprogramme oder Funktionsmodule, die zum überwiegenden Teil auf den universellen Mikrorechnern AMS-D2/D3 ablauffähig sind, zeichnen sich durch strikte Trennung von Daten und Programmcode aus. Steuertabellen vereinfachen den modularen Softwareentwurf und schaffen eine flexible Handhabbarkeit bei späteren Änderungen.

10. BETRIEBSSYSTEM

Die Komplexität der Aufgaben, ihre hohe geforderte Interaktionsfähigkeit und der unbedingte Zwang, durch Modularisierung zu einer überschaubaren Programmstruktur zu gelangen, machte den Einsatz eines Betriebssystems erforderlich.

Seit Dykstra die Semaphore /19/ einführte, wurden große Anstrengungen unternommen, kleine atomare Basisoperationen ("Primitives") zur Kommunikation und Synchronisation quasi parallel ablaufender sequentieller Prozesse zu definieren /20/21/22/23/. Basierend auf diesen Grundsatzuntersuchungen wurden einige Multi Tasking Executives für verschiedene Mikrorechner implementiert /24/25/26/. Diese sind zum Einsatz in kleinen, dedizierten Einprozessorsystemen vorgesehen und stellen keine Möglichkeit zur Programmentwicklung zur Verfügung. Ein solches System ist RMX80 /27/, das als Basissoftwaremodul zur weiteren Systementwicklung gewählt wurde, weil es zu Beginn des Projektes das einzige industriell erhältliche System war, das auf den gewählten Mikrorechnern, allerdings nur als Monoprozessor-Version, lauffähig ist.

Ein Programmsystem, das um ein "Multi Tasking Executive" organisiert ist, besteht aus mehreren "Tasks", deren Ausführung flexibel durch Zuteilung der Zentraleinheit von der Executive bestimmt wird. Dadurch scheinen die Tasks simultan abzulaufen, obwohl natürlich zu einer Zeit nur eine Task exekutiert werden kann. Jede Task führt nur eine bestimmte Funktion aus, so daß diese relativ klein und überschaubar gehalten werden kann und sich so ein sehr modulares Gesamtsystem ergibt. Die Kommunikation der einzelnen Task untereinander geschieht über genau festgelegte Schnittstellen. Dieses Interface läßt den Mikrorechner für das Anwenderprogramm als eine virtuelle Maschine erscheinen, die die vielen Details der Alarm-Behandlung und des sicheren Taskwechsels vor dem Anwender verbirgt. Dadurch wird die Anwenderprogrammierung stark vereinfacht, was zu

übersichtlichen, sicheren und leicht zu testenden Systemen führt.

Die durch ein Multi-Tasking Executive mögliche quasi-parallel Abarbeitung verschiedener Programme, kann durch den Einsatz eines Multiprozessorsystems zu einer tatsächlich gleichzeitigen Ausführung von Tasks erweitert werden. Die logische Gleichzeitigkeit wird damit zur physischen.

In einem MULTIBUS/AMS System kann ein Multiprozessorsystem leicht durch Installation zweier oder mehrerer Einkarten-Rechner aufgebaut werden. Durch eine entsprechende Hardware Logik (Arbitration) wird der konfliktfreie Buszugriff gesteuert. Auch wenn jedes Board sein eigenes Betriebssystem hält, ist damit jedoch noch keine Intertask-Kommunikation auf verschiedenen Prozessoren ohne weiteres möglich. Um ein Softwarepaket nicht nur auf einem, sondern gleichzeitig auf mehreren Prozessoren zu exekutieren, müssen zusätzliche Mechanismen zur Interprozessor-Intertask-Kommunikation bereitgestellt werden.

Mehrprozessor-Echtzeit-Betriebssysteme wurden im Hochschul- und Forschungsbereich entwickelt /28/29/, auch einige industrielle Implementierungen sind inzwischen bekannt /30/31/32/. Für die verwendeten Einkartenrechner war zu Beginn des Projektes jedoch kein Multiprozessor-Betriebssystem erhältlich. Deshalb wurde der RMX80 Executive von Intel für Mehrprozessoranwendungen erweitert. Dazu wurde eine transparente, einheitliche Schnittstelle zur Intertask-Kommunikation festgelegt, die den Anwendungsprogrammen die komplexe Umgebung des Multiprozessorsystems verbirgt. Auch dieses Interface stellt wiederum dem Anwender eine virtuelle Maschine zur Verfügung, die in diesem Fall die Leistungsfähigkeit mehrerer Zentraleinheiten bietet.

Für kleinere Systeme sind die Möglichkeiten, die ein Multitasking Executive zur Verfügung stellt, meist genügend. Intelligentere Systemfunktionen führen in der Regel zu einem hohen Speicherverbrauch. Andererseits führt die Einfachheit

der gebotenen Systemdienste zu einer hohen Flexibilität. Aber gerade diese kann den Aufbau eines strukturierten größeren, insbesondere eines verteilten Systems empfindlich erschweren. Um auch größere Systeme handhabbar zu machen, sind zusätzliche Strukturierungsvorschriften und die Definition höherer Abstraktionslevel notwendig.

Ein effizientes Strukturierungshilfsmittel ist ein Mehrebenensystem, das dem Anwender erlaubt, die Ebenen als Interface zu seinen Programmen zu wählen, die er jeweils benötigt /33/34/. Kleine Systeme benötigen meist nur die unterste Ebene, die Multi-Tasking Executive, die natürlich ihrerseits wieder ebenenorientiert aufgebaut sein kann. Größere Systeme können durch eine weitere Ebene, das "I/O Interface" bedient werden, das in Kapitel 10.3 als einer der Schwerpunkte der Entwicklungsarbeiten bei dem hier vorgestellten Projekt näher beschrieben wird. Bei noch komplexeren Systemen sind weitere höhere Abstraktionsebenen denkbar.

Zum Verständnis der im Rahmen des Entwicklungsvorhabens implementierten Betriebssystemsfunktionen (Multiprozessor Erweiterung (vgl. Kapitel 10.2), geräteunabhängige Ein/ Ausgabe (vgl. Kapitel 10.3)) werden im folgenden die wesentlichen Eigenschaften des RMX80 Systems beschrieben.

10.1 Monoprozessor Echtzeit-Multitasking Executive

Das verwendete RMX80-System der Fa. Intel /27/ ist zum Einsatz auf den MULTIBUS 8080/85 Einplatinenrechnern entwickelt worden, es kann jedoch auch bei entsprechender Auslegung der Hardware auf eigenentwickelten Mikrorechnerboards betrieben werden. Es besteht aus einem kleinen, aber sehr effizienten Systemkern (Nucleus, circa 2K Bytes) und einer Reihe von Erweiterungen, die die Anwenderprogrammierung und den Test des Gesamtprogrammsystems entscheidend erleichtern können. Es läßt sich modular konfigurieren, so daß der Anwender wahlweise die Systemteile integrieren kann, die für seine Anwendung notwendig sind. Der Prozeß

des Auswählens und Zusammenstellens der gewünschten Module wird in einer neuen Version des RMX80 Betriebssystem mit einem interaktiven Konfigurierungsprogramm auf einem Entwicklungssystem vorgenommen. Die verfügbaren RMX80 Systemmodule sind in Abb. 10.1-1 darstellt.

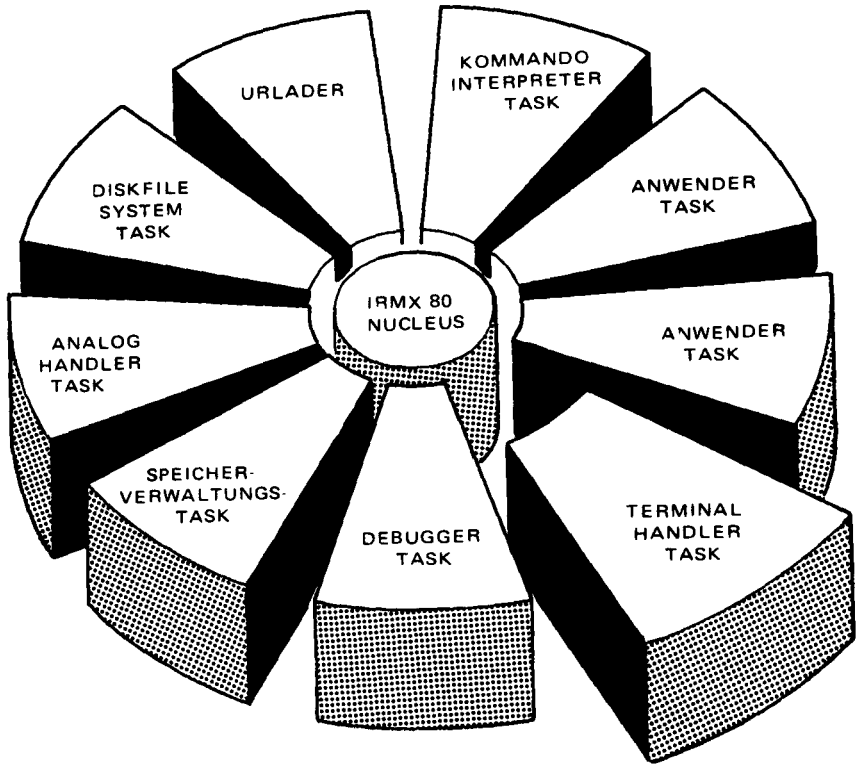


Abb. 10.1-1 Modulare Struktur des RMX80 Systems /27/.

10.1.1 Betriebssystemkern

Der Nucleus stellt einen Satz elementarer Betriebssystem-Funktionen zur Verfügung zur Steuerung, Überwachung und Synchronisierung mehrerer quasi parallel auszuführender Aufgaben (Tasks) (ein sehr ähnliches System wird von der Firma Siemens unter dem Produktnamen SMP-RTOS für das SMP80 Kartensystem /10/ vertrieben). Im wesentlichen organisiert der Nucleus die

- Task Ausführung
- Intertask Kommunikation und Synchronisation
- externe Interrupts.

Die Anzahl der Tasks ist grundsätzlich nur durch den zur Verfügung stehenden Speicherplatz begrenzt. Jede Task besitzt normalerweise ihren eigenen Programm-, Datenbereich und Stack. Die Kennzeichnung für die Wichtigkeit einer Task ist die Priorität. Im RMX80 System werden bis zu 256 Prioritäten vergeben.

Der Basis Synchronisations- und Kommunikationsmechanismus der Tasks ist die Botschaft (Message). Eine Task, die Informationen zu einer anderen senden will, schickt eine Message zu einer Exchange bzw. zu einer Mailbox. Eine Botschaft ist einfach ein zusammenhängender Datenbereich im Speicher. Die einzige Festlegung, die durch den Systemkern gefordert wird, ist der Botschaftszeiger in den ersten 2 Bytes zur Reihung der Botschaft an der Mailbox (vgl. Abb. 10.1.2-2).

Eine Task, die Daten benötigt, wartet an der Mailbox, bis eine Botschaft eintrifft bzw. bis eine spezifizierte Zeit (Time Out, Time Delay) abgelaufen ist (vgl. Abb. 10.1.1-1).

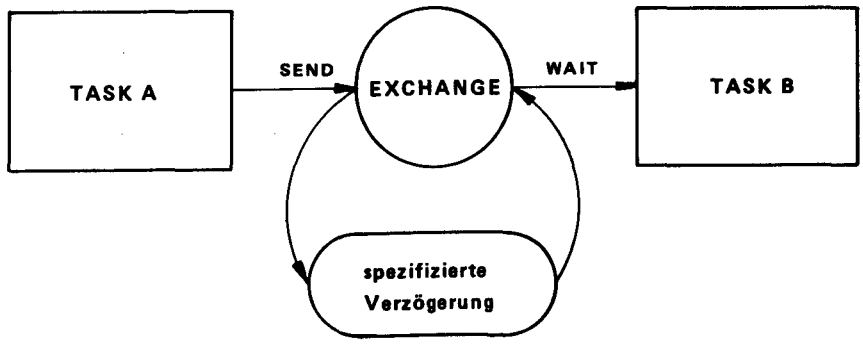


Abb. 10.1.1-1 Basis Kommunkationsmechanismus des RMX80 Betriebssystems über Briefkästen, die auch "Exchanges" oder "Mailboxes" genannt werden.

Weil mehrere Botschaften zu einer Task gesendet werden können (bzw. mehrere Tasks an einer Mailbox warten können),

hat jede Mailbox eine Warteschlange für Botschaften und Tasks. Es ist jedoch prinzipiell unmöglich, daß Tasks und Botschaften zugleich warten. Eine der beiden Warteschlangen wird immer leer sein. Die Inter-Task-Kommunikation über die Mailbox läuft demzufolge bezüglich der Botschaften nach dem FIFO Prinzip (first in/first out) und bezüglich der Tasks nach dem FCFS Prinzip (first come/first served) ab.

Eine wartende Task muß eine der folgenden Festlegungen treffen:

- Ich benötige die Botschaft unbedingt und werde deshalb so lange wie nötig warten (indefinitive wait)

- Ich wünsche die Botschaft, aber ich warte nur eine maximale Zeit von festzulegenden Systemzeiteinheiten (timed wait)

- Ich wünsche zwar die Botschaft, aber ich bin nicht bereit zu warten; wenn keine Botschaft vorhanden ist, werde ich sofort das Programm fortsetzen.

Die Leistungsfähigkeit des RMX80 Nucleus wird wesentlich durch die Konstruktion der Exchanges (Mailboxes) bestimmt. Es können grundsätzlich drei fundamentale Mechanismen unterschieden werden, mit deren Hilfe Tasks miteinander in Beziehung treten:

- Kommunikation
  Eine Task sendet einer anderen eine Botschaft über eine Mailbox.

- Synchronisation
  Eine Task A muß Aktivitäten erledigen, ehe eine Task B in ihrem Ablauf fortfahren kann. Task B beginnt dazu ein "indefinitive wait" an einer Mailbox. Wenn A ihre Aktivität beendet hat, sendet sie eine Botschaft zu dieser Mailbox, wo Task B diese erhält und bereit zur Ausführung wird. Dadurch werden die Aktivitäten der beiden Tasks miteinander synchronisiert.

- Mutual Exclusion (gegenseitiger Ausschluß)
  Der gegenseitige Ausschluß kann z.B. der Übergabe gemeinsamer Datenbereiche mehrerer Tasks dienen; z.B. kann eine Task A, die in einen Datenbereich schreibt, von einer anderen mit höherer Priorität verdrängt werden, die aus demselben Datenbereich liest. Dadurch können durch die evtl. erst teilweise Beschreibung des gemeinsamen Datenbereiches bei der Abarbeitung der Task B Fehler entstehen. Deshalb müssen Datenbereiche (wie auch andere Resourcen), die gemeinsam von mehreren Task verwendet werden, entsprechend geschützt werden. Über eine reservierte Mailbox können zur Lösung dieses Problems vereinbarte Botschaften ausgetauscht werden, deren Auswertung den gegenseitigen Ausschluß garantiert. Nur die Task, welche die vereinbarte Botschaft (Token) besitzt, kann auf den vereinbarten Datenbereich zugreifen.

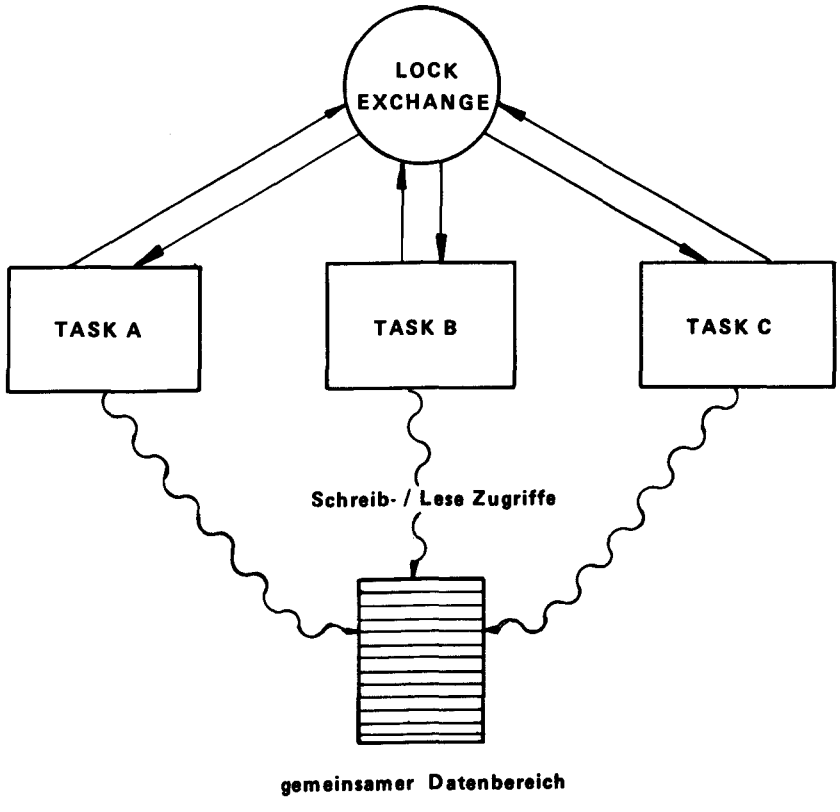


Abb. 10.1.1-2 "Mutual Exclusion" Mechanismus durch Auswertung eines "TOKEN" über eine "LOCK EXCHANGE"

Eine der wichtigsten Anforderungen an ein Echtzeitbetriebssystem ist die Fähigkeit auf externe Ereignisse reagieren zu können. Diese (Interrupts, Alarme) werden für den Systemkern sichtbar, wenn ein elektrischer Impuls an einem Interrupteingang des Prozessors eintrifft. RMX80 sendet dann eine Botschaft zu einer speziellen Interrupt Exchange, wo eine hoch priore Task wartet, um auf das auftretende Ereignis zu reagieren. Dadurch wird die Alarmbedienung ebenso behandelt wie der Austausch von Botschaften zwischen Hintergrund-Tasks.

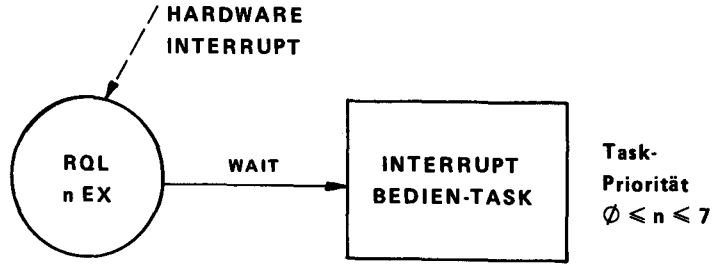


Abb. 10.1.1-3 Alarmbehandlung in einem RMX80 Betriebssystem

Jede Task hat neben anderen Attributen, wie z.B. der Priorität, zu jeder Zeit nur einen bestimmten Zustand im System:

- "bereit" (ready)
  Eine bereite Task ist rechenbereit und wartet auf Zuteilung der Zentraleinheit durch den Betriebssystemkern.

- "rechnend" (running)
  Die rechnende Task besitzt die Kontrolle über die Zentraleinheit und wird ausgeführt.

- "wartend" (waiting)
  Eine Task in diesem Zustand wartet an einer Exchange auf eine Botschaft oder den Ablauf eines spezifizierten Zeitintervalls oder beides.

- "suspendiert" (suspended)
  Eine Task kann sich selbst oder eine andere Task suspendieren. Sie bleibt in diesem Zustand, bis sie von einer anderen Task weitergeführt wird. Die suspendierte Task bewirbt sich nicht um Rechenzeit auf dem Prozessor.

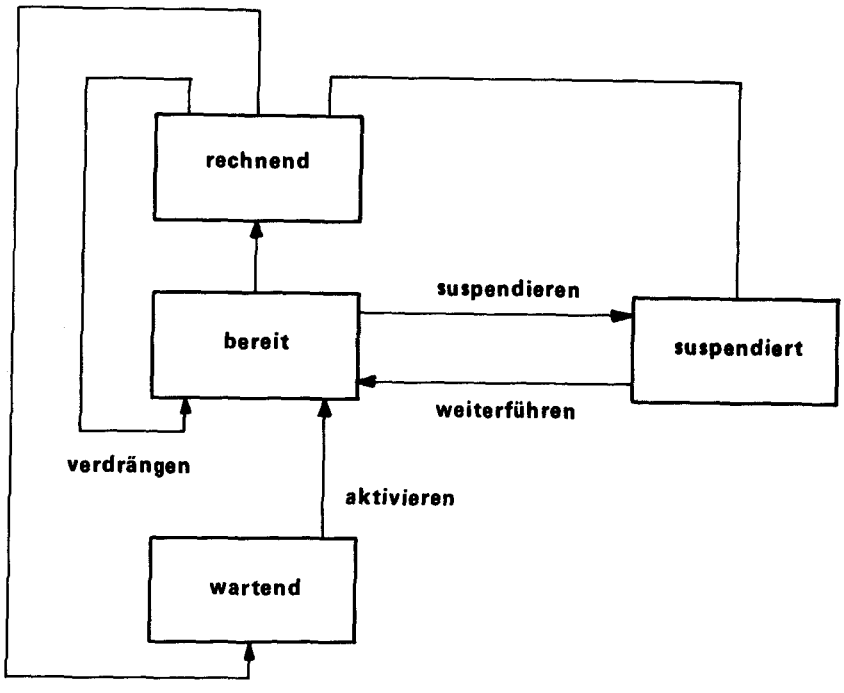


Abb. 10.1.1-4 Task Zustandsdiagramm

Der Zuweisungsmechanismus (scheduling algorithm), der die rechnende Task bestimmt, nimmt Bezug auf die Priorität und den Zustand der Task. Es gilt immer: zur rechnenden Task wird die bereite Task mit der höchsten Priorität. Diese Art des Schedulings ist bekannt als prioritätsbezogenes Verdrängungs-Scheduling, d.h. eine höher priore Task, die bereit wird, verdrängt immer eine rechnende niedrigerer Priorität. Der Scheduler tritt immer dann in Aktion, wenn folgende Ereignisse auftreten:

- Hardware Interrupts
- Aufrufe bestimmter Systemdienste durch eine Task

10.1.2 RMX80 Basisoperationen

Tasks können vom Systemkern Dienste anfordern. Im wesentlichen werden vom Nucleus folgende Basisoperationen zur Verfügung gestellt:

- Sende eine Botschaft zu einer Mailbox (RQSEND)
- Warte auf eine Botschaft an einer Exchange (RQWAIT)
- Suspendiere eine Task (RQSUSP)
- Führe eine Task weiter (RQRESM)
- Richte eine Task ein (RQCTSK)
- Lösche eine Task (RQDTSK)
- Generiere eine Exchange (RQCXCH)
- Lösche eine Exchange (RQDXCH)

Wenn eine Task A einer Task B eine Botschaft senden will, geschieht dies wie oben beschrieben über eine Exchange. Dabei soll die Variable MSGADR die Adresse der Message von Task A enthalten; die Variable XCHADR enthält die Adresse der Exchange selbst (beiden Tasks bekannt). Task A sendet die Botschaft mit dem Systemaufruf

CALL RQSEND (XCHADR, MSGADR).

Task B kann MSGADR und damit die Botschaft selbst durch den Aufruf

MESSAGE = RQWAIT (XCHADR, TIME OUT)

erhalten.

Vor dem Aussenden einer Botschaft mit Hilfe der Systemfunktion RQSEND muß die sendende Task die Botschaft in einem der empfangenden Task zugänglichen Speicherbereich aufbauen. Jede Botschaft setzt sich aus einem 5 Byte "Botschaftskopf" und einem "Botschaftsrumpf" von beliebiger Länge zusammen. Das allgemeine Format einer Botschaft ist in Abb. 10.1.2-1 dargestellt. Der Systemkern erzeugt die Daten für die ersten beiden Bytes der Botschaft (Bot-

schaftszeiger), er dient der Verkettung von Botschaften für das Betriebssystem und darf nicht von der Anwendertask verändert werden.

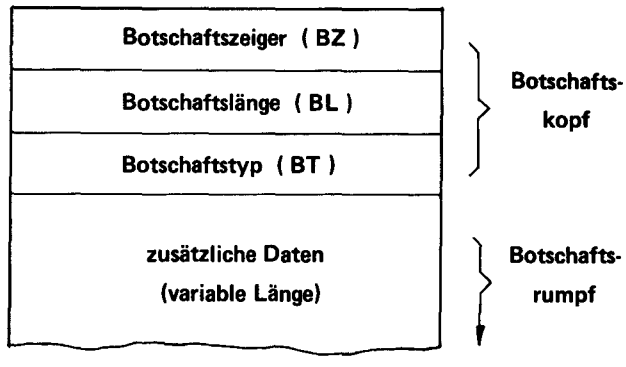


Abb. 10.1.2-1 Aufbau einer Botschaft im RMX80 Betriebssystem

An einer Exchange können sowohl Botschaften als auch Tasks in eine Warteschlange eingereiht werden. Dies geschieht mit Hilfe eines Mailbox Kontrollblocks, dessen prinzipieller Aufbau in Abb. 10.1.2-2 dargestellt ist.

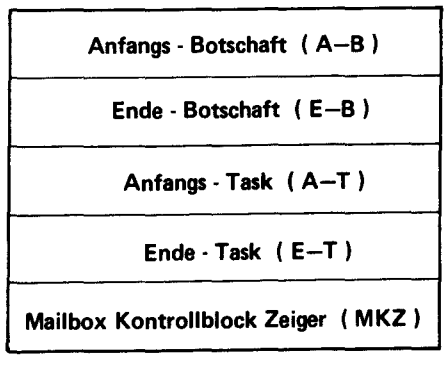


Abb. 10.1.2-2 Aufbau Mailbox-Kontrollblock

Durch Verkettung läßt sich nach Abb. 10.1.2-3 eine Botschaftswarteschlange aufbauen.

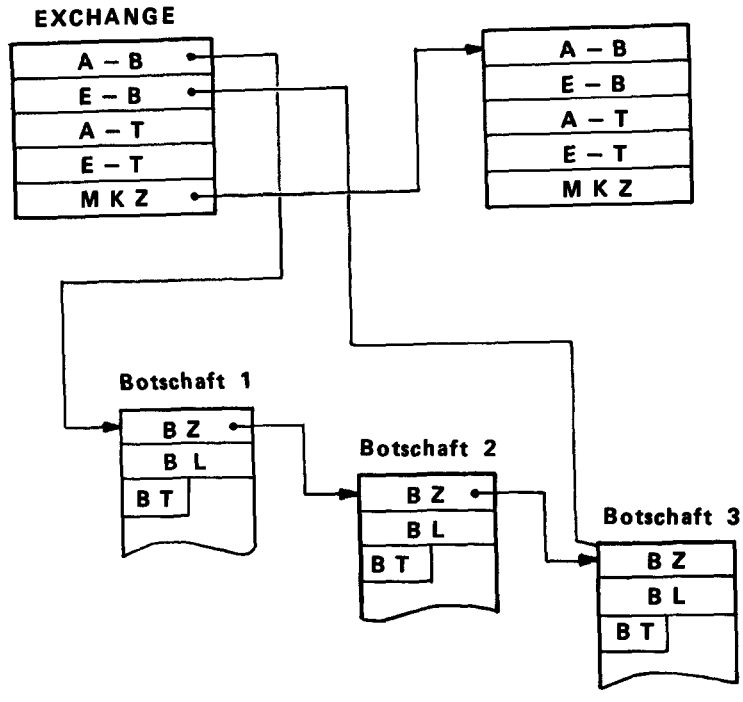


Abb. 10.1.2-3 Botschaftswarteschlange an einer Exchange

10.1.3 RMX80 Systemmodule

Zur Vervollständigung der Systembeschreibung werden im folgenden die weiteren Module des RMX80 Executives kurz genannt (eine weitere, tiefer führende Erklärung gibt /35/ (vgl. auch Abb. 10.1-1)):

- Speicherverwaltung (Free Space Manager).
  Die Speicherverwaltung ermöglicht die dynamische Zuweisung von Speicherbereichen während der Ausführung der Programme. Sie kann von den Tasks durch Aufruf der Systemfunktionen

  - Anforderung Speicherplatz
  - Freigabe Speicherplatz

  aktiviert werden. Anforderungen können sich auf jede

Größe eines zusammenhängenen Speicherbereichs beziehen, der dann nach Verfügbarkeit zugeteilt wird.

- Plattenverwaltungssystem (Disk File System)
  Das Disk File System erlaubt den Echtzeitzugriff auf einen Floppy-Massenspeicher, es sind die bekannten Systemaufrufe OPEN, READ, WRITE, SEEK, CLOSE, RENAME und DELETE implementiert.

- Terminal Handler
  Der Terminal Handler stellt einen Datenpfad zwischen einem angeschlossenen Sichtgerät und den einzelnen Anwendertasks her, die vom Scheduler des Nucleus verwaltet werden. Die Kommunikation zwischen dem Peripheriegerät und den einzelnen Tasks geschieht über die Systemdienste SEND und WAIT.

- Kommando Interpreter (Command Line Interpreter)
  Er akzeptiert Kommandos von der Systemkonsole und aktiviert die zugehörigen Behandlungsroutinen der Anwender Programme.

- Testhilfen (Debugger)
  Das Testprogramm wird normalerweise nur während der Entwicklungsphase in das System geladen. Es gestattet das Setzen von Halte-Punkten in einer Task, Prüfung und Modifizieren von Speicherinhalten und den Ausdruck interner Systemlisten.

- Bootstrap
  Der Bootstrap Loader erlaubt das Laden eines RMX80 Systems von der Platte.

- Analog Treiber (Analog Handler)
  Der Analog Treiber arbeitet zusammen mit Analog zu Digital bzw. Digital zu Analog Convertern der Firma Intel, die multibuskompatibel sind.

10.2 Multiprozessor Erweiterungen

Im nachfolgenden wird ein Satz von einfachen Erweiterungen des RMX80 Executives beschrieben, die die Multiprozessor-Kommunikation über einen gemeinsam benutzten Speicher erlaubt. Bei der Auslegung der Erweiterungen mußten die folgenden Randbedingungen beachtet werden:

1. Da kein Quellcode des RMX80 Systems vorlag, konnten keine internen Modifikationen des Systems vorgenommen werden.

2. Die Multiprozessor-Kommunikation sollte so transparent wie möglich sein. Es sollte für eine Task kein Unterschied bestehen, ob ihr Kommunikationspartner sich auf derselben Rechnerplatine oder einer beliebigen anderen befindet. Damit wird der Programmcode einer Task völlig unabhängig von der physikalischen Konfiguration des Systems.

Jeder Prozessor in dem beschriebenen System verfügt über eine eigene RMX80 System Kopie. Der Basis-Kommunikationsmechanismus ist damit die Message. Interprozessor-Botschaften müssen demzufolge in einem allen Prozessoren gemeinsam zugänglichen Speicher abgelegt werden.

Eines der Hauptprobleme in einem Multiprozessorsystem liegt in der Festlegung der Adresse einer Exchange in einem anderen Prozessor. Eine Möglichkeit wäre, alle Interprozessor-Exchanges PUBLIC zu deklarieren in dem System, wo sie existieren und sie EXTERNAL zu deklarieren für alle anderen Systeme. Diese Implementationsvariante hat einige Nachteile, deren wesentliche in der Tatsache liegt, daß bei jeder Änderung eines Programmsystems alle anderen Systeme auch geändert werden müssen, die Botschaften dorthin senden (dabei ist eine EPROM Änderung unumgänglich), um jede Änderung der Interprozessor-Mailbox-Adressen zu berücksichtigen. Es wird also ein Verfahren benötigt, daß eine "Remote Exchange" identifizieren kann, ohne die Kenntnis ihrer Adresse in dem Zielprozessor.

Dies geschieht hier durch eine Dispatch-Tabelle, die jedem Prozessor zugeordnet wird und aus einem Block von bis zu 255 Zeigern auf lokale Exchanges besteht. Diese dem jeweiligen System zugehörigen Exchanges können so - indirekt - von anderen Prozessoren identifiziert über einen Index angesprochen werden. Dadurch wird eine "Remote Exchange" durch die Vergabe eines Zeigers in der Dispatch-Tabelle des Zielprozessors eindeutig festgelegt. Durch neue Kompilierläufe werden die Zeiger nicht verändert; der Eintrag in die Tabelle wird während der Konfigurierung vorgenommen.

Zwei Attribute sind nötig, um in einem Multiprozessorsystem eine Exchange eindeutig zu identifizieren: Eines ist der oben beschriebene Eintrag in die Dispatch-Tabelle, ein anderes die Prozessor-Nummer, auf dem die Tabelle implementiert ist. Jedem Prozessor ist deshalb eine eindeutige Nummer zugeordnet. Der "Ziel-Identifikator" (REMOTE IDENTIFIER) einer Exchange ist somit ein Zwei-Byte-Wort mit dem Exchange-Zeiger (Index) und der Prozessoradresse. Diese Informationseinheit läßt ein eindeutiges Auffinden einer Exchange innerhalb des Gesamtsystems zu (vgl. Abb. 10.2-1).

Der Ziel-Identifikator hat damit dasselbe Format wie eine Exchange-Adresse und kann deshalb in den normalen SEND und WAIT Aufrufen substituiert werden. Tatsächlich werden in diesem Multiprozessor-System Botschaften nicht zu Exchanges, sondern zu Ziel-Identifikatoren geschickt.

Jeder der n Prozessoren hat festgelegte Adressen (Mailboxes) im gemeinsamen Speicher, über die er Zeiger zu Botschaften von anderen Prozessoren empfängt, die für Exchanges in seinem eigenen RMX-System bestimmt sind.

Abb. 10.2-1 erklärt das Senden einer Botschaft von Prozessor A (Source) zu Prozessor B (Destination). Zunächst verschafft sich Prozessor A einen exklusiven Zugang zur oben beschriebenen Mailbox des Prozessors B im gemeinsamen Speicher. Dies geschieht über eine unteilbare Test und Set Operation auf die entsprechende Mailbox Semaphore. Dann

lädt der Prozessor die Mailbox mit der Adresse der Botschaft und dem Zeiger (Dispatch Tabelle) der Exchange in Prozessor B, zu der die Botschaft gesendet werden soll. Danach aktiviert Prozessor A den Prozessor B über einen Interrupt. Die Interrupt Service Task wertet den Inhalt der Mailbox im gemeinsamen RAM aus und sendet die Botschaft zu der Exchange durch Aufruf der normalen Systemfunktion RQSEND. Um die Botschaft zu empfangen, führt eine Task in Prozessor B eine RQWAIT Funktion auf die Ziel-Exchange aus.

RQSEND und RQWAIT sind eingebettet in die beschriebenen Funktionen. Um eine Botschaft zu senden, führt eine Task grundsätzlich den Systemaufruf

```
CALL MPSEND (DESTINATION, MESSAGE $ ADDR)
DESTINATION =   REMOTE $ IDENTIFIER  /  EXCHANGE
```

aus. Wenn DESTINATION eine RMX80 Exchange Adresse auf demselben Board ist, wird RQSEND aufgerufen, um die Botschaft im eigenen System zu transferieren. Wenn es ein Ziel-Identifikator ist, wird die Botschaft mit Hilfe des oben beschriebenen Mailbox/Interrupt Mechanismus zu dem entsprechenden Prozessor gesendet.

Um eine Botschaft zu empfangen, ruft die entsprechende Task die folgende Systemfunktion auf:

```
MESSAGE $ ADDR = MPWAIT (DESTINATION, DELAY)
```

Wenn DESTINATION ein Ziel-Identifikator in der Dispatch Tabelle ist, wartet die Task an der spezifizierten Exchange. Diese Exchange kann nur lokal sein.

Exchange-Zeiger können dynamisch in die Dispatch-Tabelle ein- bzw. ausgetragen werden unter Verwendung der beiden Systemfunktionen MPCXCH und MPDXCH, welche die normalen Aufrufe RQCXCH und RQDXCH ersetzen. Der einzige Unterschied ist, daß MPCXCH ein Prozeduraufruf ist, der einen 2 Byte Wert liefert, den Ziel-Identifikator der Exchange. Dies ist

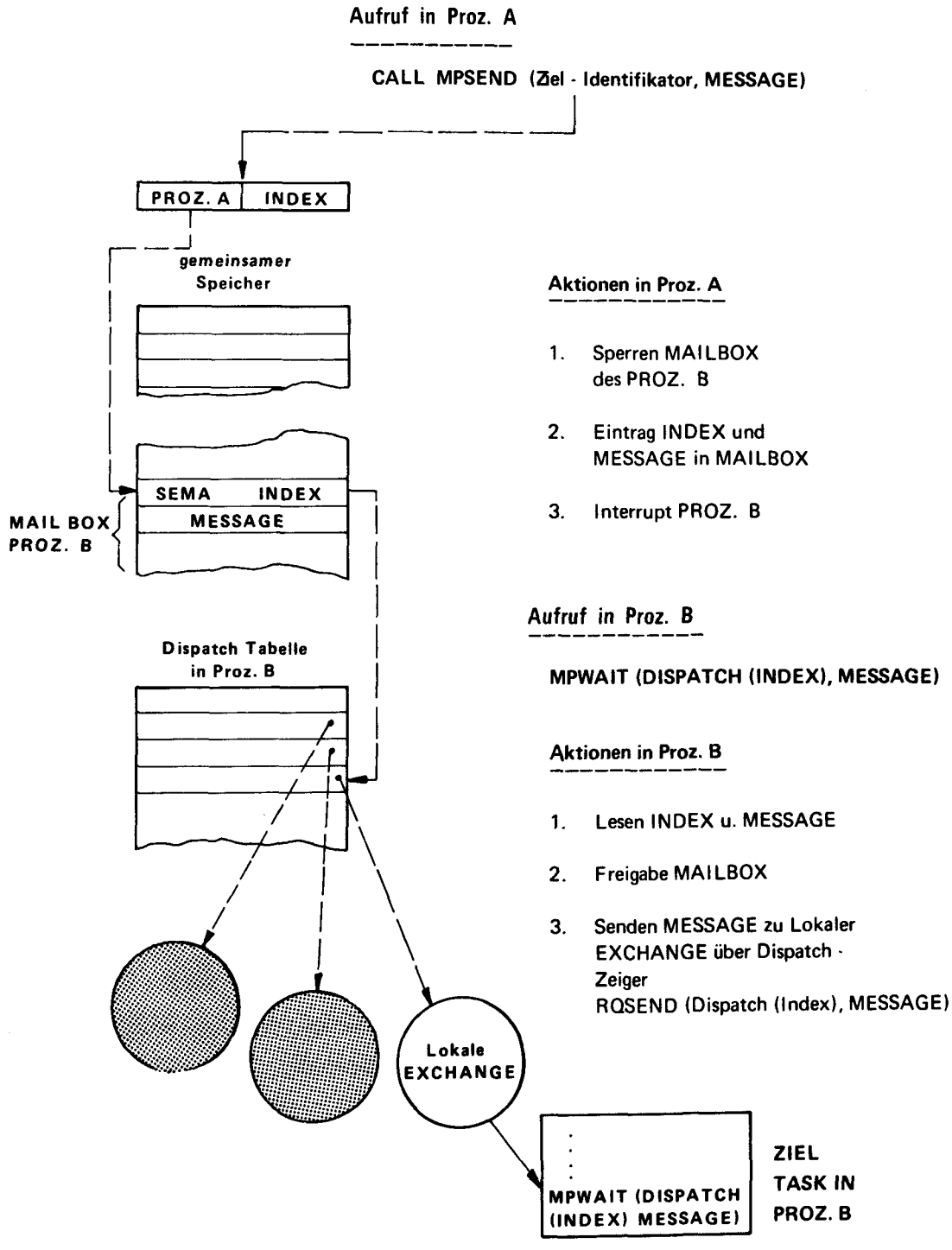


Abb. 10.2-1 Senden einer Botschaft von Prozessor A nach Prozessor B

der einzige nicht transparente Systemaufruf im Rahmen der Multiprozessor-Erweiterung. In einem Ein-Prozessorsystem wird eine Dummy-Version vorgesehen, die einfach die Exchange Adresse zurücküberträgt, die ihr übergeben wurde.

MPDXCH verhält sich ebenso wie der Aufruf RQDXCH, abgesehen von der Tatsache, daß er Ziel-Identifikatoren und tatsächliche Exchange Adressen als Eingangsparameter akzeptiert. Weiterhin kann der Aufruf nur auf lokale Exchanges angewendet werden.

10.2.1 Speicherverwaltung des gemeinsamen RAM

Die Interprozessor-Kommunikation macht intensiven Gebrauch von dem gemeinsam benutzten Speicher. Die statische Zuweisung von Speicherbereichen zu einzelnen Prozessoren während des Kompilier- bzw. Linkvorganges ist eine sehr schlechte Lösung, die das dynamische Wachsen des Multiprozessorsystems nur ungenügend berücksichtigen kann.

Eine komfortable, dynamische Speicherzuweisung läßt sich mit Hilfe des RMX80 Free Space Manager erreichen, indem der gemeinsame Speicherbereich als "buffer pool" von diesem verwaltet wird. Eine Task, die einen gemeinsamen Speicherbereich benutzen will, fordert diesen bei der Speicherverwaltung an. Dadurch werden Speicherbereiche nur belegt, wenn sie auch tatsächlich benötigt werden.

Das Speicherverwaltungssystem wird auf einem der Prozessoren installiert. Um die Speicheranforderungen der einzelnen Prozessoren dabei zu gewährleisten, muß ein fester Speicherbereich für die Anforderungsbotschaften festgelegt werden. Deshalb wurde die oben besprochene Mailbox auf 128 Bytes erweitert, obwohl für die Interprozessorkommunikation nur 4 Bytes benötigt wurden (vgl. Abb. 10.2-1).

10.3 Geräteunabhängige Ein/Ausgabe

In Kapitel 9 wurde die Ebenenstruktur des hier verwendeten (und teilweise eigens entwickelten) Betriebssystems dargestellt. Ein weit verbreitetes und sehr nützliches Strukturierungskonzept genügender Abstraktion ist die dabei gezeigte geräteunabhängige Ein/Ausgabe-Ebene des Systems (device independent I/O), die dem Anwendungsprogrammierer

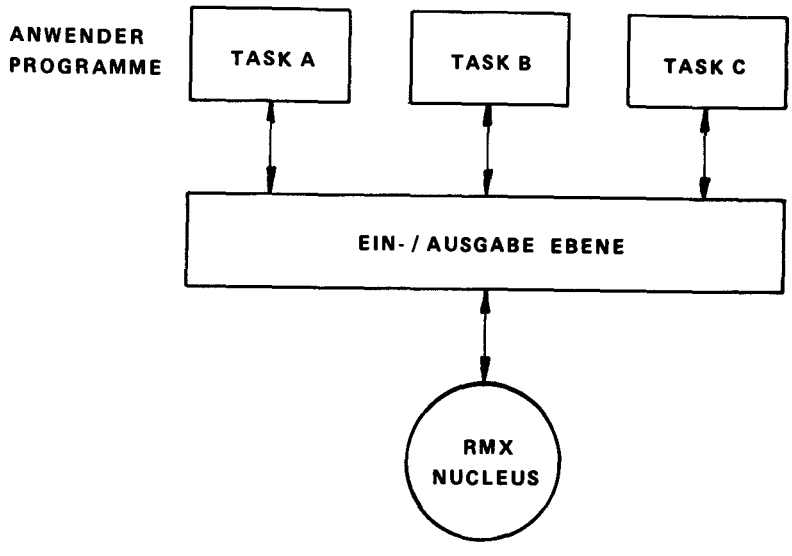


Abb. 10.3-1 Einheitliche Schnittstelle zum Betriebssystem über die geräteunabhängige Ein/Ausgabe-Ebene

eine einheitliche Schnittstelle zur Außenwelt bietet. Dabei bleiben ihm die Details der Intertask Kommunikation und Synchronisation der tieferen Schichten des Betriebssystems und die speziellen Eigenarten des angesprochenen Peripheriegerätes verborgen (vgl. auch /13/).

Dadurch können Anwenderprogramme implementiert werden, die ganz oder zumindest zum überwiegenden Teil unabhängig von den angesprochenen Geräten sind. Dieses Konzept ist bekannt und ein Schlüsselelement in den meisten hochwertigen Mini- bzw. Mikrorechnerbetriebssystemen. Die hier beschriebene Implementationsart wird auch bei UNIX /36/, ISIS-II,

(Intel) /37/ und mit einigen Einschränkungen bei CP/M (Digital Research) /38/ angewendet.

Diese Systeme organisieren die gesamte Ein/Ausgabe mittels "logischer Kanäle", die die Verbindung von den Programmen zu den peripheren Geräten herstellen. Eine direkte Kommunikation ist nur in Ausnahmefällen zugelassen.

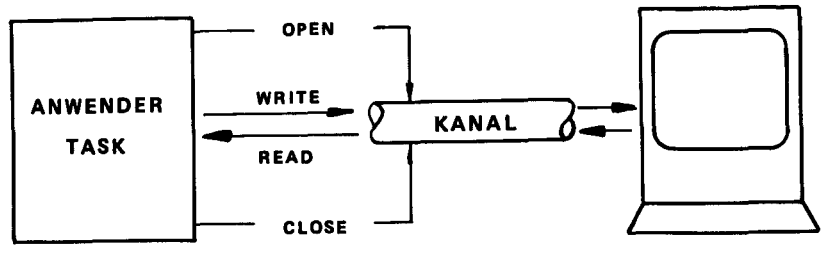


Abb. 10.3-2 Kommunikation zwischen Anwenderprogramm und Peripheriegerät über einen logischen Kanal.

Das Konzept der geräteunabhängigen Ein/Ausgabe läßt sich durch die folgenden Eigenschaften charakterisieren:

- Alle Geräte verwenden einheitliche logische Kanäle zum Betriebssystem, dadurch erscheinen sie identisch für das Anwenderprogramm.

- Ein Gerätetreiber paßt die individuelle Eigenschaft eines Gerätes an die Kanalschnittstelle an. Dieser besteht aus mehreren Tasks oder Prozeduren, die den eigentlichen Datentransfer zwischen der einheitlichen Kanalschnittstelle und dem mitunter sehr speziellen Gerät besorgen.

- Ein vereinbarter, gemeinsamer Satz von Befehlen wird von den Gerätetreibern bereitgestellt, die die Basis Ein/Ausgabefunktionen übernehmen (logische Kanaloperationen).

- Die Kanalkonstruktion ist üblicherweise dynamisch, d.h. sie wird nur für die Zeit des Datentransfers dynamisch kreiert.

In den oben zitierten Betriebssystemen ist die beschriebene Systemarchitektur beschränkt auf die physikalische Geräteunabhängigkeit (z.B. auf Terminals, Lochstreifenleser und Stanzer usw.) und das File System.

Einen höheren Grad von Abstraktion erreicht man, wenn man als Gerät jede Senke bzw. Quelle von Daten innerhalb eines Systems zuläßt. Diese "logischen Geräte" werden dann vom Anwenderprogramm ebenso angesprochen wie physikalisch real vorhandene Peripheriegeräte. Dadurch wird der logische Kanal nichts anderes als ein Satz von Schnittstellenvereinbarungen. Jede Task, die diese Vereinbarung befolgt, kann als Gerätetreiber angesehen werden und kann deshalb über die logischen Kanaloperationen angesprochen werden. In einem verteilten System z.B. kann die Kommunikation zwischen Tasks auf verschiedenen Prozessoren über logische Kanäle abgewickelt werden, wobei jede Task die andere als Gerät betrachtet (d.h. als Senke bzw. Quelle von Daten).

10.3.1 Kanal Operationen

Die Schnittstelle zwischen Programm (Task) und Kanal kann durch die anwendbaren Operationen hinreichend beschrieben werden, die das Programm hinsichtlich des Kanals ausführen kann.

In dem beschriebenen System sind 4 Basis-Kanaloperationen vorgesehen:

OPEN
READ
WRITE
CLOSE

OPEN: Um einen Kanal aufzubauen, führt das Programm einen OPEN Befehl aus. Das Programm spezifiziert über einen Namen das Gerät, mit dem es kommunizieren will und alle anderen für das spezielle Gerät relevanten Informationen. Die Ein/Ausgabe Ebene des Betriebssystems (vgl. Abb. 10.3-3) stellt den Kanal zur Verfügung und informiert den zuständigen Gerätetreiber, der das angeschlossene Gerät aktiviert und der Ein/Ausgabe-Ebene Status bzw. Fehlermeldungen übergibt. Konnte der Kanal erfolgreich geöffnet werden, wird dem aufrufenden Programm die zugeteilte Kanalnummer für die gewünschte Transaktion übergeben. Diese Nummer genügt den nachfolgenden Operationen zur eindeutigen Identifizierung des Gerätes.

READ: Die Lese-Operation transferiert Daten vom aufgerufenen Gerät zum Programm. Als Parameter für die Ein/Ausgabe-Ebene des Betriebssystems sind dazu notwendig: die Kanalnummer, der Beginn des Datenbereiches, in den die gelesenen Daten abzulegen sind und die maximale Datenmenge, die empfangen werden kann. Der Gerätetreiber übergibt dem Programm die transferierten Daten bzw. meldet einen aufgetretenen Fehler.

WRITE: Die Schreib-Operation läuft ebenso wie das Lesen ab, jedoch mit entgegengesetzter Transferrichtung.

CLOSE: Nachdem ein Programm seinen Datentransfer mit einem Gerät abgeschlossen hat, kann es die Kanalverbindung über eine CLOSE-Operation abbauen, indem es die Kanalnummer spezifiziert. Dabei ist wiederum die Rückgabe von Fehlerbedingungen möglich.

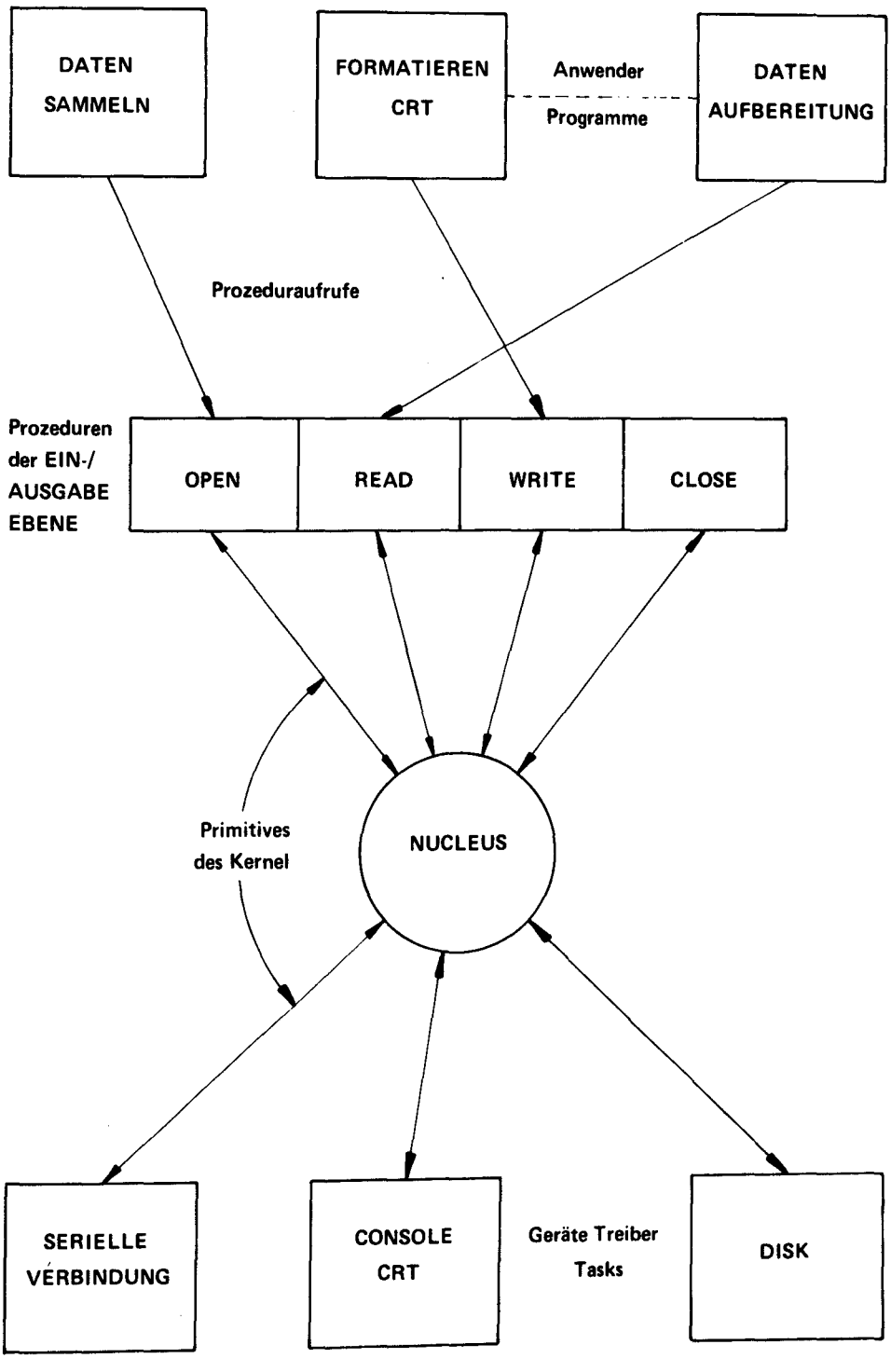


Abb. 10.3-3 Zusammenwirken zwischen Anwendertasks, Ein/ Ausgabe-Ebene, Nucleus und Gerätetreibern

Wenn einmal eine Kanalverbindung aufgebaut wurde, wird diese während aller Datentransfers nur noch durch die vergebene Nummer vom Programm her angesprochen. Es gibt keine weiteren identifizierenden Daten. Das Lesen von einer Platte ist dann für das Programm dasselbe wie das Empfangen von Daten vom Terminal bzw. vom seriellen Datenübertragungssystem, das in der Pilotinstrumentierung die Zentrale mit den einzelnen Prüfstandssteuereinheiten verbindet. Insofern sind alle Ein/Ausgabe-Operationen für die Programme "geräteunabhängig".

Die Anwendertasks sprechen mit der Ein/Ausgabe-Ebene des Betriebssystems über einen Satz von Prozeduraufrufen. Die Ein/Ausgabe-Ebene ihrerseits kommuniziert mit den Gerätetreibern über die Kommunikations- bzw. Synchronisationsmechanismen des darunterliegenden Multi-Tasking Executives.

10.3.2 Realisierte Implementation

Das Lesen eines Datenbereiches von der Platte im zentralen Rechner soll als Beispiel für die "Kanal Ein/Ausgabe" im folgenden näher beschrieben werden. Dazu soll zunächst untersucht werden, wie dieses Problem unter Verwendung der Standard RMX80-Funktionen gelöst wird.

In dem Disk File System (DFS) des RMX80 muß zunächst eine OPEN Operation durchgeführt werden, indem eine OPEN Anforderungsbotschaft zu einer Exchange mit dem Namen RQOPNX gesendet wird. Das DFS sendet daraufhin eine Botschaft zu einer vom Anwender definierten Antwort-Exchange, an der die anfordernde Task wartet. In dieser Botschaft ist dann vom DFS die Adresse einer weiteren Exchange spezifiziert, zu der alle folgenden Anforderungen für den gewünschten Datenbereich gesendet werden müssen.

Zu dieser Mailbox wird daraufhin der eigentliche Lesebefehl geschickt. Die anfordernde Task wartet dann an ihrer Antwort-Exchange und das DFS schickt nach erfolgtem Daten-

transfer dorthin eine Information über die Länge des tatsächlich gelesenen Datenbereiches. Wenn die Anwendertask die gewünschten Daten vollständig erhalten hat, sendet sie eine CLOSE Anforderung an das DFS. Das in PLM codierte Beispiel in Abb. 10.3.2-1 illustriert diesen Vorgang im einzelnen.

Die Operationen des DFS orientieren sich in ihrer Struktur an den beschriebenen Kanalaufrufen und dienten für die dargestellte Implementation als ein Modell zur Entwicklung der Ein/Ausgabe-Ebene. Diese besteht aus vier PUBLIC Prozeduren, die die Kanal-Operationen implementieren. Sie sind reentrant programmiert und können deshalb zu jeder Zeit von jeder Task aufgerufen werden.

Um einen Kanal aufzubauen, führt eine Task den Aufruf

```
CALL OPEN (CHNLPNTR, NAMEPNTR, PARAM, STATUSPNTR)
```

aus. Die Parameter sind dabei die folgenden:

1. Adresse der Variablen, wo die Kanalnummer niedergelegt werden soll.

2. Zeiger zu einem "file name block", ein String von ASCII Text bestehend aus:

   : dev : name . ext del

   dev ist zwei Buchstaben lang und spezifiziert das Gerät, das ausgewählt wird. Die Interpretation von name und ext sind geräteabhängig. del ist ein Delimiter z.B. CR, Ø, der das Ende des "file name blocks" anzeigt.

3. Ein geräteabhängiger Parameter. Normalerweise repräsentiert er eine Zugriffsart. Er kann auch einen Zeiger zu einem Block mehrerer Parameter bilden.

4. Adresse einer Variablen, unter der die OPEN-Prozedur jede Art von Fehlermeldungen anzeigt.

Zum Lesen von Daten (READ) führt eine Anwendertask den folgenden Aufruf aus:

CALL READ (CHANNEL, BUFFERPNTR, COUNT, ACTUALPNTR, STATUSPNTR)

mit den Parametern:

1. Kanalnummer, wie sie von der OPEN Operation bereitgestellt wurde.

2. Speicheradresse eines Puffers, in den die gelesenen Daten transferiert werden sollen.

3. Länge des Puffers oder Zahl der erwarteten Bytes.

4. Adresse einer Variablen, in die READ die Zahl der gelesenen Bytes transferiert.

5. Adresse einer Variablen, unter der die Fehlermeldungen abgelegt werden.

Die Schreiboperation ist ähnlich dem Leseaufruf und hat dieselbe Parameterbedeutung:

CALL WRITE (CHANNEL, BUFFERPNTR, COUNT, ACTUALPNTR, STATUSPNTR)

Der Abbau der Kanalverbindung wird über CALL CLOSE (CHANNEL, STATUSPNTR) vorgenommen. Die zugehörigen Parameter sind oben beschrieben. In Abb. 10.3.2-2 ist das Lesen des Datenbereiches dargestellt unter Verwendung der Prozeduren der Ein/Ausgabe-Ebene des Betriebssystems. Beim Vergleich der in Abb. 10.3.2-1 und Abb. 10.3.2-2 dargestellten Implementationsvarianten ist leicht zu erkennen, daß die Benutzung der entwickelten geräteunabhängigen Ebene eine

kürzere und leichter zu verstehende Programmversion zur Folge hat.

```
/* Declare exchanges.  EXCHANGE$DISCRIPTOR is a literal
    symbol which declares the appropriate data structure. */
    DECLARE RQOPNX EXCHANGE$DESCRIPTOR EXTERNAL;
    DECLARE RESP$EX EXCHANGE$DESCRIPTOR PUBLIC;

/* External procedures */
      RQSEND: PROCEDURE (EXCH$PTR,MSG$PTR) EXTERNAL;
       DECLARE (EXCH$PTR,MSG$PTR) ADDRESS;
      END RQSEND;

      RQWAIT: PROCEDURE (EXCH$PTR,DELAY) EXTERNAL;
       DECLARE (EXCH$PTR,DELAY) ADDRESS;
      END RQWAIT;

/* Message structures. OPEN$REQUEST$MSG and READ$REQUEST$MSG
    are literal symbols for declaring the appropriate data
    structures. The two message use the same memory space. */
      DECLARE OPEN$MSG OPEN$REQUEST$MSG;
      DECLARE READ$MSG READ$REQUEST$MSG AT (.OPEN$MSG);

/* Local variables */
      DECLARE (DUMMY,AFR$XCH) ADDRESS;
      DECLARE BUFFER(256) BYTE;
      DECLARE FILE$NAME(*) BYTE DATA(':F1:HISTOG.DAT',0);

/* Set up Open Request Message and send it */
      OPEN$MSG.TYPE = OPEN$REQUEST$TYPE;
      OPEN$MSG.RESPONSE$EXCHANGE = .RESP$EX;
      OPEN$MSG.FILE$PTR = .FILE$NAME;
      OPEN$MSG.ACCESS = 1;  /* Read access */

      CALL RQSEND (.RQOPNX,.OPEN$MSG);
      DUMMY = RQWAIT (.RESP$EX,0);  /* wait forever */

/* Set up for read operation */
      AFR$XCH = OPEN$MSG.AFR$XCH;
      READ$MSG.TYPE = READ$REQUEST$TYPE;
      READ$MSG.BUFFER = .BUFFER;
      READ$MSG.COUNT = 256;
      READ$MSG.ACTUAL = NOT 0;

      IF OPEN$MSG.STATUS <> 0 THEN DO;
                Error recovery
      END;
      ELSE DO WHLIE READ$MSG.ACTUAL <> 0;

/* End of file is indicated by ACTUAL = 0 */
        CALL RQSEND (AFR$XCH,.READ$MSG);
        DUMMY = RQWAIT (.RESP$EX,0);

        IF READ$MSG.STATUS <> 0 THEN DO;
                Error recovery
        END;
        ELSE DO;
                Process data
        END;
      END;

/* Set up ard send Close request */
      READ$MSG.TYPE = CLOSE$REQUEST$TYPE;
      CALL RQSEND (AFR$XCH,.READ$MSG);
      DUMMY = RQWAIT (.RESP$EX,0);
```

Abb. 10.3.2-1 Lesen eines Platten-Datenbereiches unter Verwendung der RMX80 Kommunikationsprozeduren

```
/* Declare external procedures */
      OPEN: PROCEDURE (CHNL$PTR,FILE$PTR,PARAM,STS$PTR) EXTERNAL;
        DECLARE (CHNL$PTR,FILE$PTR,PARAM,STS$PTR) ADDRESS;
      END OPEN;

      READ: PROCEDURE (CHNL,BUF$PTR,COUNT,ACT$PTR,STS$PTR) EXTERNAL;
        DECLARE (CHNL,BUF$PTR,COUNT,ACT$PTR,STS$PTR) ADDRESS;
      END READ;

/* Declare local variables */
      DECLARE (IN$FILE,ACTUAL,STATUS) ADDRESS;
      DECLARE BUFFER(256) BYTE;

/* Open a file on floppy disk drive ':F1:' for reading */
      CALL OPEN (.IN$FILE,.(':F1:HISTOG.DAT',0),1,.STATUS);
      ACTUAL = NOT 0  /* Initialise */
      IF STATUS <> 0 THEN DO;
              Error recovery
      END;
      ELSE DO WHILE ACTUAL <> 0;

/* End of file is indicated by ACTUAL = 0  */
      CALL READ (IN$FILE,.BUFFER,256,.ACTUAL,.STATUS);
      IF STATUS <> 0 THEN DO;
              Error recovery
      END;
      ELSE DO;
              Process data
      END;
    END;

    CALL CLOSE (IN$FILE,.STATUS);
```

Abb. 10.3.2-2 Lesen eines Platten-Datenbereiches mit Hilfe der Prozeduren der entwickelten Ein/Ausgabe Ebene des Betriebssystems.

10.3.3 Interne System Operationen

Jede Prozedur der beschriebenen Betriebssystemebene bildet eine Botschaft aus den ihr übergebenen Parametern und sendet sie zu der zugehörigen Exchange, die sie mit dem entsprechenden Gerätetreiber verbindet. Dann wartet sie an einer Antwort-Exchange auf die Reaktion des Gerätetreibers und übergibt die Antwortparameter (STATUS und ACTUAL) zum aufgerufenen Programm. Jeder Kanal hat seinen eigenen Anforderungs- und Antwort-Briefkasten (Exchange), der für alle Operationen benutzt wird.

Ein Gerätetreiber in diesem System besteht aus mehreren Tasks, in der Regel zwei: einer "Initiator Task", die die OPEN Operation behandelt und einer "Transfer Task", die alle anderen Kanaloperationen ausführt. Die "Initiator Task" wartet an ihrer zugeordneten Exchange ("OPEN Exchange"). Die Adresse der OPEN Exchange und der zwei Buchstaben lange Gerätename sind für jedes Gerät in einer Geräteliste niedergelegt. Wenn die OPEN Prozedur aufgerufen wird, prüft sie den empfangenen Namen gegen die Einträge in der Geräteliste ab. Bei Übereinstimmung sendet sie die Anforderungsbotschaft für den gewünschten Kanal zu der

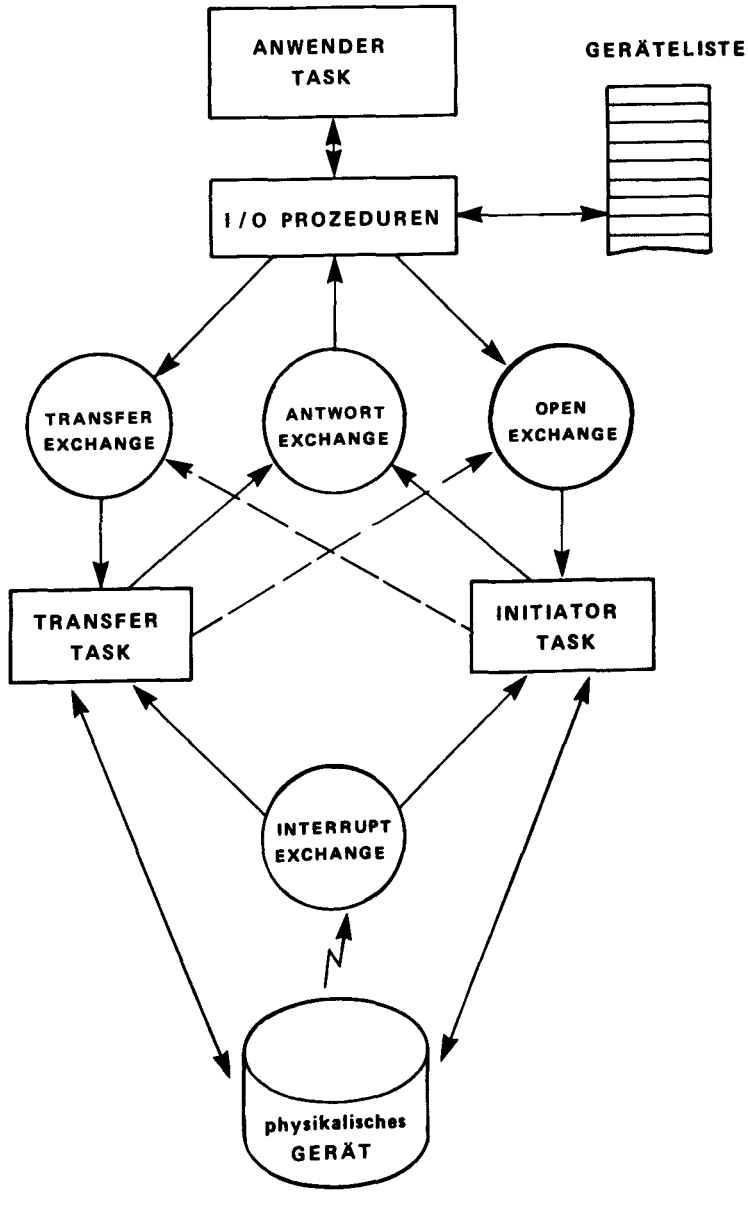


Abb. 10.3.3-1 Interner Mechanismus der I/O Prozedur-Aufrufe

zugehörigen OPEN Exchange als eine OPEN Anforderung. Nach der Aktivierung des Gerätes sendet die "Initiator Task" über die Antwort Exchange dieses Kanals eine Quittung mit der Adresse einer "Transfer Exchange", über die alle weiteren Anforderungen für den aufgebauten Kanal laufen (vgl. Abb. 10.3.3-1).

Die Transfer Task wartet an diesem Briefkasten auf Lese-, Schreib- und "CLOSE"-Aufträge. Nach Bedienung einer Anforderung setzt sie die ACTUAL und STATUS Felder der Anforderungsbotschaft und sendet sie zu der Antwort Mailbox des Kanals zurück.

10.3.4 Gerätetreiber in verteilten Systemen

Die Anwendung, für die das System entwickelt wurde, ist ein verteiltes System mit 3 Knoten und einem zentralen Supervisor, die über serielle Verbindungsleitungen miteinander verkehren (vgl. Abb. 6.2). Die Software des Datenübertragungssystems ist als Gerätetreiber innerhalb der Ein/Ausgabe-Ebene des Betriebssystems organisiert. Dadurch wird die Kommunikation zu jedem abgesetzten Knotenrechnersystem ebenso behandelt wie die Ein/Ausgabe zu den lokalen Peripheriegeräten. Dazu wird ein logischer Kanal aufgebaut zwischen einer Task im Supervisor und einer Task in einer der Prüfstandsteuerungen. Die Tasks müssen dazu wechselseitig eine OPEN Operation anfordern und dabei den Namen der anderen Task in dem "file name block" spezifizieren. Danach sehen die Tasks sich gegenseitig als "File". Ein Datentransfer geschieht immer dann, wenn eine Task eine READ Operation und die andere eine entsprechende WRITE Operation ausführt. Eine detaillierte Darstellung der Kommunikationsabläufe wird in /13/ gegeben. Zur Abwicklung der Kommunikationsfunktionen werden im Zentralrechnersystem und den einzelnen Prüfstandssteuereinheiten autonome Mikrorechnerkarten eingesetzt. Ihre Aufgaben sind:

- Generierung der untersten Protokollebene

- Datensicherung, Übertragungsfehlererkennung und Korrektur
- Empfangen, Senden und Aufbereiten der Nachrichten für beide Übertragungsrichtungen
- Verwalten von mehreren Nachrichten, die in Warteschlangen eingereiht werden.

Das Datenübertragungssystem als logische Geräteeinheit bildet die Basis für die gemeinsame Benutzung der Resourcen in der Zentrale von den einzelnen Knotenrechnersystemen (remote resource sharing) aus. Zum Beispiel hat eine Task, die auf einem der Prozessorsysteme in den Prüfstandssteuereinheiten abläuft, Zugriff auf den Massenspeicher in der Zentrale, als wenn sie selbst ein lokales Plattensystem zur Verfügung hätte. Dies wird durch die Definition einer "logischen Platten-Geräteeinheit" in dem abgesetzten Rechnersystem erreicht, die ihrerseits wieder die logische Geräteeinheit "Datenübertragungssystem" mitbenutzt, um mit

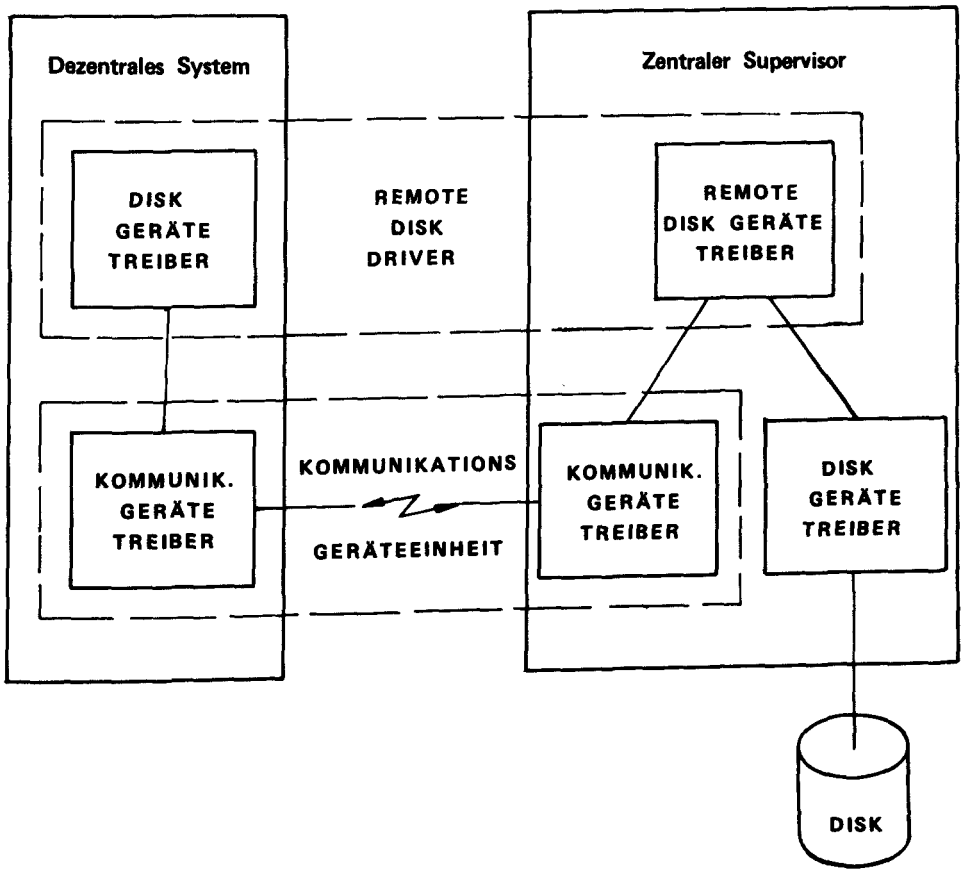


Abb. 10.3.4-1 Verteilte Geräteeinheiten

einer Task in der Zentrale zu verkehren. Erst diese greift dann auf Anforderung der entfernten Task über den entsprechenden Gerätetreiber auf das physikalisch existente Speichermedium zu. Von der Konzeption ist ein solcher Gerätetreiber "verteilt" über das gesamte System nach Abb. 10.3.4-1.

Der Anwendungsprogrammierer braucht keine Rücksicht darauf zu nehmen, wo sich das angesprochene Gerät tatsächlich befindet. Er hat Zugriff auf logische Geräte, die auf jedem Prozessorboard repräsentiert sind durch ein Element des "verteilten Gerätetreibers".

Die bis jetzt beschriebenen Kanalaufrufe laufen synchron ab; d.h. eine Prozedur geht nur zur aufrufenden Task zurück, wenn die verlangte Operation beendet ist. In einem Monoprozessorsystem führt dies nicht zu einem Effiziensverlust, weil der Gerätetreiber sowieso eine gewisse Prozessorzeit braucht, um die geforderten Funktionen ausführen zu können. In einem Multiprozessorsystem dagegen könnte die aufrufende Task mit anderen Aufgaben fortfahren, während der von ihr aufgerufene Gerätetreiber auf einem anderen Prozessor abläuft.

Dieses Problem wurde durch die Definition von Erweiterungen in den Prozeduraufrufen gelöst:

```
CALL READC      (..............)
CALL WRITEC     (..............)
CALL WAITC      (..............)
```

Dabei initiieren READC und WRITEC dieselben Kanaloperationen wie READ und WRITE, nur geben sie danach die Kontrolle sofort zum aufrufenden Programm zurück (C steht für Continue). Wenn das Programm das Ergebnis der Kanaloperation benötigt, führt es den Aufruf CALL WAITC aus, der seinerseits das Programm nur weiterlaufen läßt, wenn die Operation beendet ist.

11. PROBLEMORIENTIERTE FUNKTIONSEINHEITEN UND DIENSTPROGRAMME

11.1 Ablaufsteuerung (ABL)

Das Ablaufsteuerungsmodul (ABL) ist auf einer Mikrorechnerkarte AMS-D3 aufgebaut. Es koordiniert die Aktivitäten der Prüfstandsteuereinheit. Der hierzu erforderliche Informationsaustausch mit den übrigen Teilnehmern am Multiprozessor-Bus findet über das in Kap. 10.2 beschriebene Kommunikationsverfahren statt. Die Funktionen der Ablaufsteuerung, die in Kap. 6 skizziert wurden, werden im folgenden ausführlicher beschrieben mit besonderer Berücksichtigung der Interaktionen zwischen den Tasks der ABL und denen benachbarter Module.

11.1.1 Programmaufbau der Ablaufsteuerung

Das Ablaufsteuerungsmodul wird, wie die übrigen Bus-Master, von dem Betriebssystem verwaltet.

Neben der Systemsoftware laufen auf der ABL vier Anwendertasks:

- INIT für die Systeminitialisierung
- ASYNC zur Verarbeitung asynchron eintreffender Ereignisse
- PRAS, der Prüfablauf-Sprachinterpreter
- DIGAUS, eine anwendungsorientierte Task zur digitalen Ausgabe.

In Abb. 11.1.1-1 sind Programmsystem und Kommunikationspfade der Tasks auf dem Ablaufsteuerungsmodul graphisch dargestellt.

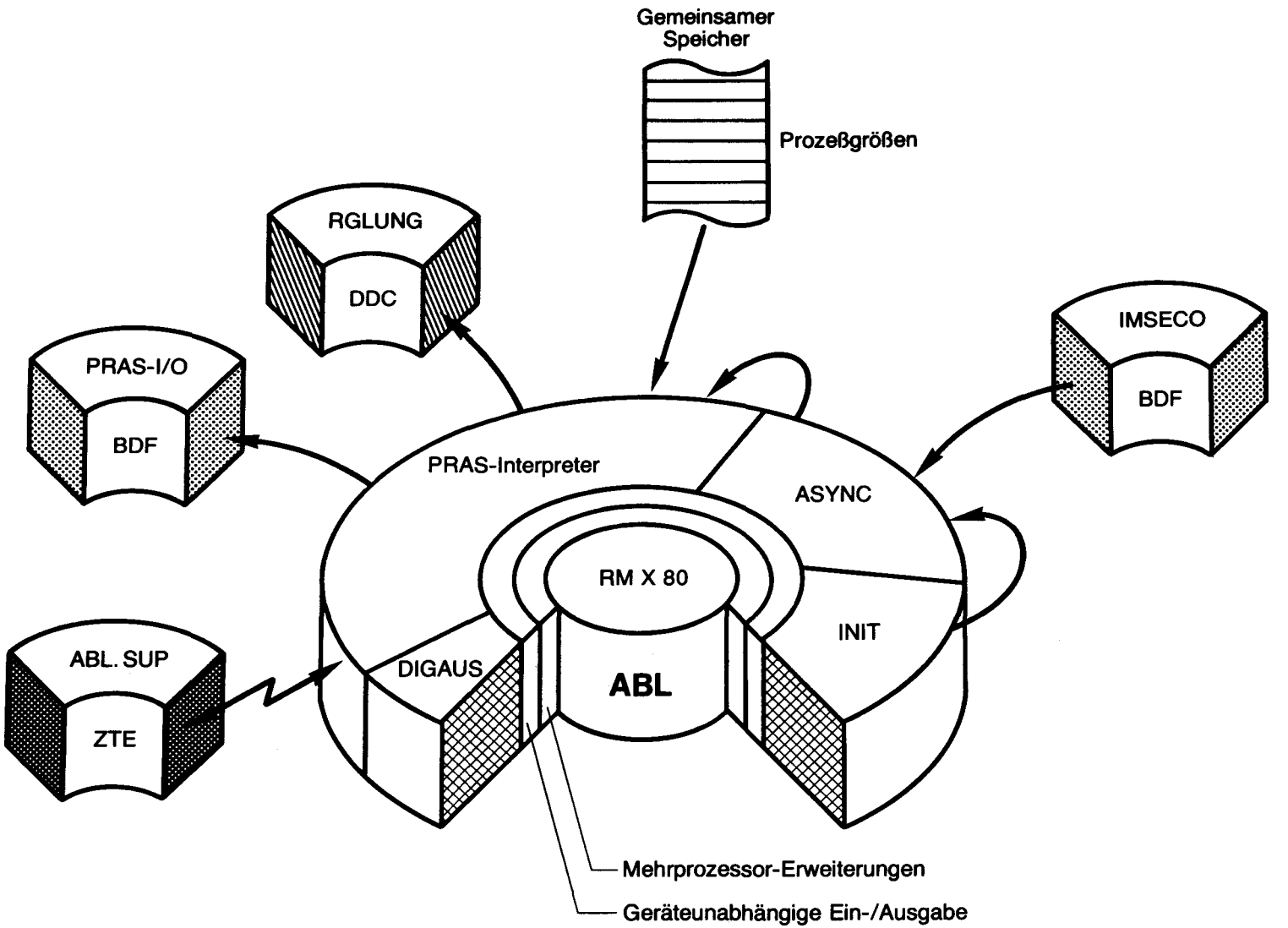


Abb. 11.1.1-1 Programmsystem Ablaufsteuerung

11.1.1.1 Initialisierung

Nach einem Systemrestart wird die Initialisierung des ABL-Moduls von der Task INIT durchgeführt. Dies umfaßt einen Selbsttest bestehend aus: EPROM-Test, RAM-Test, Test von Timerbaustein und Interrupt-Controller sowie den Funktionstests des AMS-Systembusses und des gemeinsamen Speichers. Damit wird sichergestellt, daß die hardwaremäßige Voraussetzung für eine fehlerfreie Kommunikation mit den anderen Busteilnehmern gegeben ist. Die Task INIT wartet nun auf die "Fertig"-Meldung der übrigen Module und gibt - wenn alle Module sich startbereit gemeldet haben - die Kontrolle an die Task ASYNC ab und damit das System für den Bediener frei.

11.1.1.2 Verarbeitung von Bedieneranforderungen

Nach Abschluß der Systeminitialisierung erwartet die Task ASYNC Anforderungen des Bedieners, die von der Task IMSECO der Bedienfeldverwaltung an ASYNC auf dem Ablaufsteuerungsmodul geschickt werden. Diese Anforderungen können sein:

- Bedienungsfunktionen für die Prozeßperipherie (Semi-Automatikbetrieb)
- Programmlade- und Startbefehl
- Programmabbruchbefehl.

Die eintreffenden Anforderungen werden auf Zulässigkeit und Plausibilität geprüft und eine Quittung bzw. Fehlermeldung an den Bediener zurückgegeben.

11.1.1.3 Bedienungsfunktionen für die Prozeßperipherie (Semi-Automatikbetrieb)

Mit Hilfe der Semi-Automatikfunktion kann der Bediener die prozeßorientierten Funktionen der Prüfablaufsprache PRAS

als unmittelbar auszuführende Befehle (immediate Mode) aufrufen und so den Prüfstand bedienen. Diese Befehle umfassen die Funktionen:

- Vorgabe des Sollwertes analoger Prozeßgrößen (Druck, Durchfluß, Strom, Drehzahl usw.)
- Bedienung digitale Prozeßelemente (Relais, Ventile)
- Aufruf von Prüfmakros.

Über die Bedieneranforderung können sowohl Einzelfunktionen der Prozeßperipherie als auch Prüfmakros (Conditions) aufgerufen werden, die eine sequentielle Ausführung definierter Einzelfunktionen darstellen. Der Bediener wählt die gewünschte Funktionsgruppe aus einem Bildschirmmenü, das in Abb. 11.1.1.3-1 dargestellt ist.

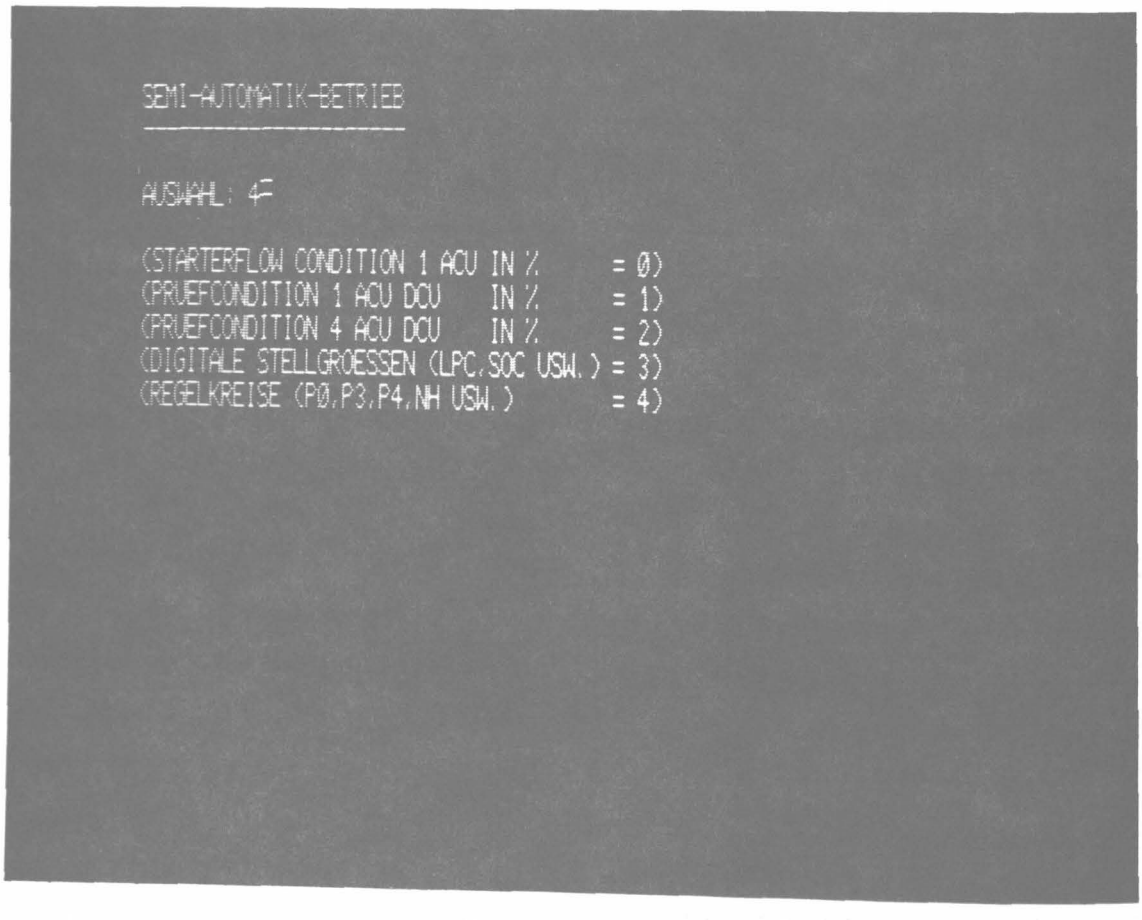


Abb. 11.1.1.3-1 Funktionsmenü für Semi-Automatikbetrieb

Als Beispiel ist die Funktionsgruppe "Regelkreise" (Analoge Prozeßgrößen) ausgewählt worden. Das Submenü erlaubt dem

Bediener jetzt den Zugriff auf die Einzelfunktionen. Über die Tastatur des Bedienfeldes werden die Sollwerte eingegeben. Die Istwerte werden auf dem Bildschirm numerisch und quasi analog als Balkendiagramm in Echtzeit dargestellt.

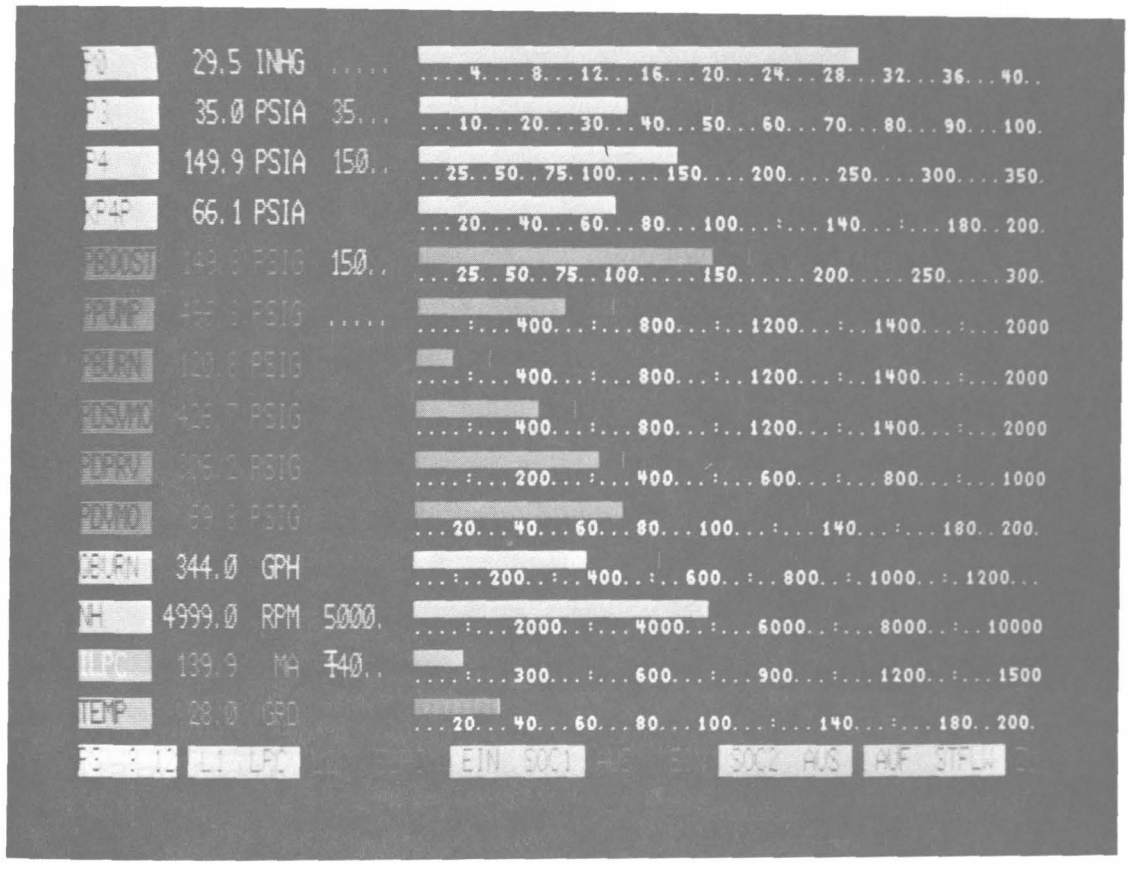


Abb. 11.1.1.3-2 Submenü zur Bedienung der analogen Prozeßgrößen mit (von links nach rechts): Prozeßgrößenbezeichnung, Istwert (numerisch), Maßeinheit, Sollwert, Istwert (quasianalog)

In einem weiteren Submenü können durch Eingabe über die Zifferntastatur des Bedienfeldes die digitalen Stellelemente der Prozeßperipherie geschaltet werden. Der Schaltzustand wird am unteren Rand des Bildschirms eingeblendet.

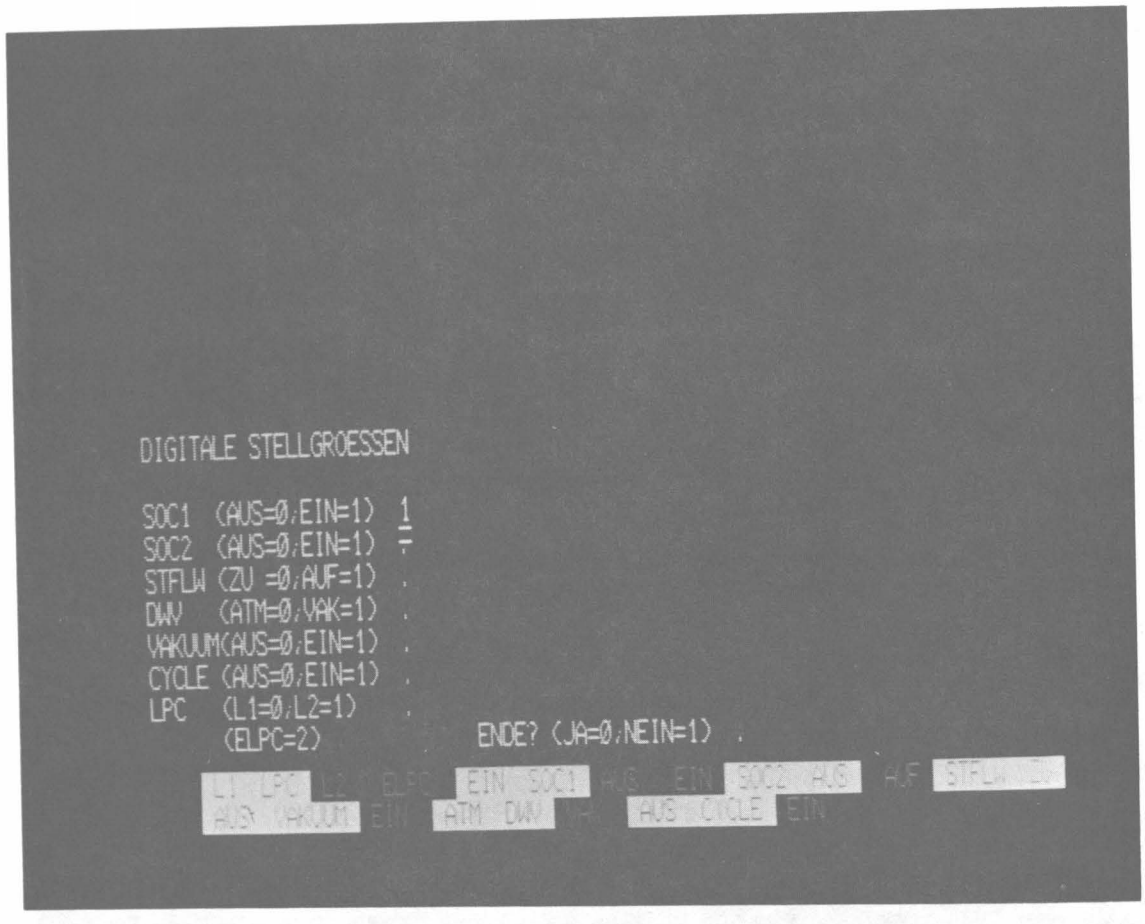


Abb. 11.1.1.3-3 Submenü zur Bedienung der digitalen Stellgrößen

Im Semi-Automatikbetrieb läuft der Prüfstand autark unter der Kontrolle der prozeßnahen Prüfstand-Steuereinheit, d.h. es können auch bei Ausfall des übergeordneten Zentralrechners die teilautomatisierten Funktionen des Prüfstandes genutzt werden.

Der Ablauf einer Befehlsausführung im Semi-Automatikbetrieb ist in Abb. 11.1.1.3-4 dargestellt.

Die Task IMSECO sendet die Bedieneranforderung (1) in Form einer direkt auszuführenden Programmzeile an den Operator-Request-Exchange OPRRQX. Die Botschaft wird von der Task ASYNC entgegengenommen (2), geprüft und der Empfang über den Response-Exchange OPRRSX quittiert (3,4). Anschließend wird die Programmzeile über den Exchange PRSRQX an den

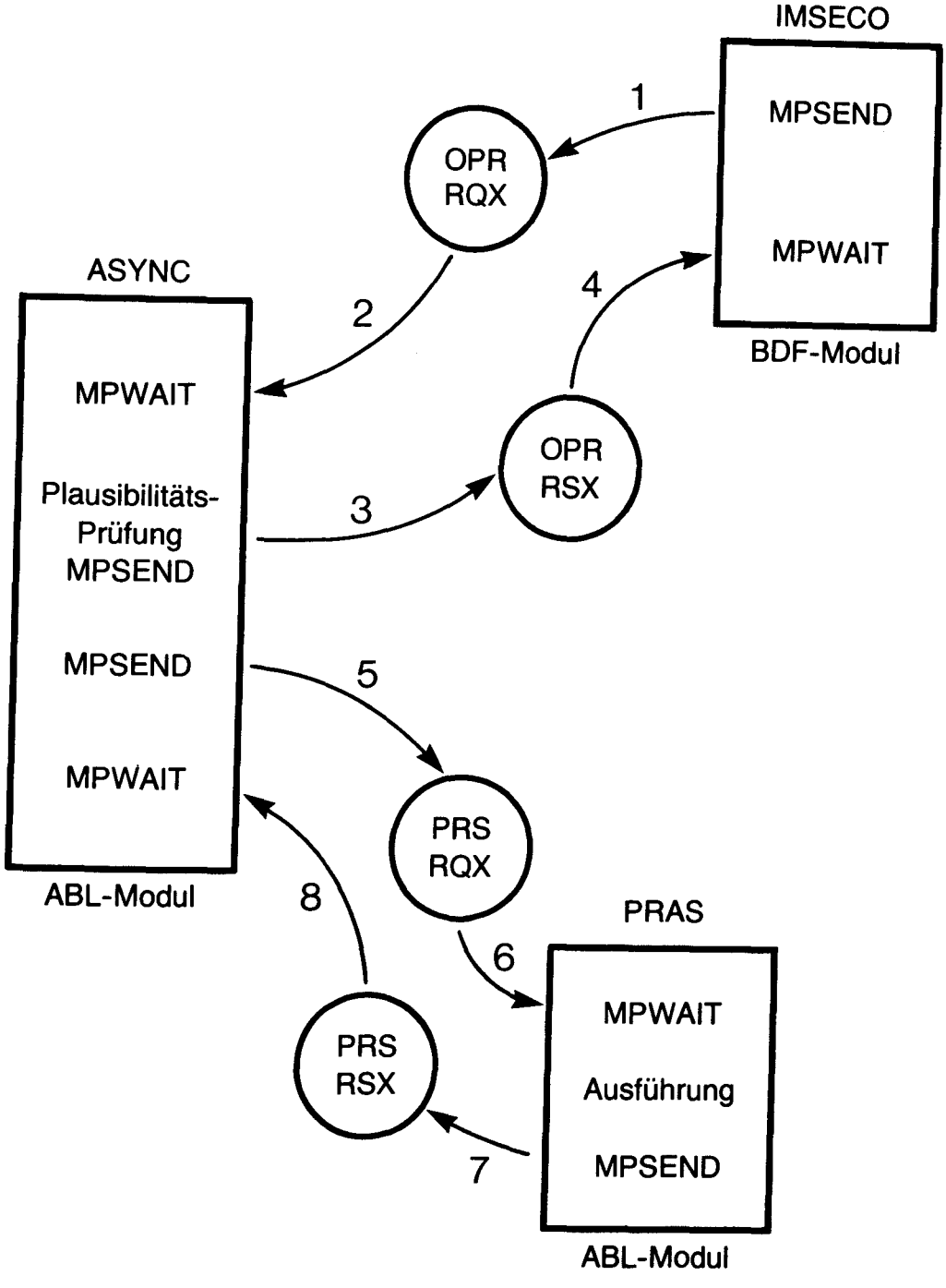


Abb. 11.1.1.3-4 Bearbeitung einer Bedieneranforderung über das Bedienfeldmodul

PRAS-Executer übergeben (5,6), der nach ihrer Ausführung eine Quittung oder ggf. eine Funktionsfehlermeldung über PRSRSX zurücksendet (7,8).

Eine detaillierte Beschreibung der Semi-Automatikfunktionen wird im Zusammenhang mit der Prüfablaufsprache PRAS in Kap. 12 gegeben.

11.1.1.4 Laden und Starten von Prüfablaufprogrammen

Auf Anforderung des Bedieners (vgl. Abb. 11.3.4-2) lädt die Ablaufsteuerung Prüfablaufprogramme von dem Floppy-Diskspeicher des Zentralrechners in den Arbeitsspeicher des Prüfsprachinterpreters PRAS. Dieser Transfer wird mit Hilfe der geräteunabhängigen Ein/Ausgabe durchgeführt, die als Service des Multiprozessor-Betriebssystems zur Verfügung steht (siehe Kap. 10.3). Die Anforderung eines Programms, das als File auf der Floppy-Disk liegt, wird über einen logischen Kanal, der bei Bedarf vom Betriebssystem bereitgestellt wird, von der Ablaufsteuerung (ABL) an eine korrespondierende Task ABL.SUP in dem Zentralrechner (ZTE) geschickt. Diese führt im Auftrag der Ablaufsteuerung die Transfers von und zur Disk durch (Remote DISK I/O). Nach erfolgreichem Laden des Prüfprogramms erhält der Bediener eine Startmeldung und die Programmausführung wird begonnen.

11.1.1.5 Programmabbruch/Ende

Ein laufendes Programm kann jederzeit vom Bediener abgebrochen werden. Dies wird bewirkt durch einen Abbruch-Befehl, der von der Task IMSECO der Bedienfeldsteuerung an die Task ASYNC geschickt wird, die ihrerseits die Programmausführung des Prüfsprachinterpreters PRAS nach Abarbeiten des aktuellen Programmstatements stoppt.

Ein Programmabbruch wird von dem Prüfsprachinterpreter selbsttätig generiert, wenn ein Programmierfehler oder Funktionsfehler der Prozeßperipherie festgestellt wird. Der Bediener wird durch eine Fehlermeldung über die Fehlerursache informiert.

Das Programmende ist das logische Ende eines Programmablaufes. Der Bediener erhält eine "Ende"-Meldung und kann durch weitere Befehlseingaben die Prüfstandfunktion steuern.

11.1.1.6 Digitale Ausgabe-Task DIGAUS

Zur effizienteren Nutzung der Hardware der Prüfstandsteuereinheit werden zur Bedienung digitaler Prozeßelemente die digitalen Ausgabeports des ABL-Moduls benutzt.

Von zwei parallelen Ausgabe-Ports werden die Steuersignale für die Bedienung von binären Prozeßelementen, wie Relais, Ventilen usw. an eine Anpaßelektronik zur galvanischen Entkopplung und Pegelanpassung geleitet, die ihrerseits über Leistungsteile die Prozeßelemente ansteuern. Der Ist-Zustand des Elements wird über einen Kontrollkontakt zurückgelesen.

Die Bedienung der digitalen Stellelemente übernimmt die Task DIGAUS auf dem ABL-Modul. DIGAUS wartet an dem Exchange DIGRQX auf Botschaften, die entweder Anforderungen des Prüfstandbedieners zur Betätigung digitaler Stellelemente sein können oder Funktionsaufrufe des Prüfsprachinterpreters aus einem Prüfablaufprogramm. DIGAUS setzt die Anforderung in einen Steuercode (Bitmuster) für die Ausgabehardware um und übergibt dieses Signal an die Peripherie. Über die Kontrollkontakte melden die peripheren digitalen Stellelemente ihren Schaltzustand zurück, aus deren Vergleich mit dem ausgegebenen Bitmuster die Funktionstüchtigkeit des Prozeßelements erkannt wird. Dem Ergebnis entsprechend wird über den Exchange DIGRSX eine "Fertig"- oder "Funktionsfehler"-Meldung an die anfordernde Task zurückgeschickt.

11.1.2 Kommunikation der Ablaufsteuerung mit den anderen Busteilnehmern

Neben dem Informationsaustausch zwischen den Tasks INIT, ASYNC und PRAS, die auf dem Ablaufsteuerungsmodul selbst angesiedelt sind, findet auch ein Datentransfer zwischen der ABL und benachbarten Modulen statt (vgl. auch Abb. 11.1.1-1).

Für diese Kommunikation werden Dienste benutzt, die die Mehrprozessor-Erweiterungen dem Betriebssystemskern RMX80 zur Verfügung stellen (siehe Kap. 10.1). Neben den bereits beschriebenen Kommunikationspfaden IMSECO - ASYNC und dem Remote Disk I/O existieren noch zwei weitere Nachbartasks, die von der ABL Daten empfangen: Steuerzeichen und Texte an die Bedienfeldausgabe PRASIO zur Bedienerführung und Sollwerte und Regelparameter an die Digitale Regelung (DDC).

Außer diesem betriebssystemunterstützten Datentransfer wird in einem einzelnen Fall eine vereinfachte hardwareflaggesteuerte Kommunikationsprozedur zum Zugriff auf die vom DDC-Modul im gemeinsamen Speicher abgelegten Prozeßgrößen (Ist-Werte) verwendet. Die Ausnahme ist notwendig, da über den Botschaftsmechanismus des Betriebssystems stets nur eine Botschaft an einen einzelnen Empfänger geschickt werden kann, die Ist-Werte der Prozeßgrößen aber allen Busteilnehmern uneingeschränkt zur Verfügung stehen sollen.

11.2 Digitale Regelung (DDC)

Die Funktionseinheit "Digitale Regelung" ist als eine ausgelagerte Funktion der Programmiersprache PRAS zu betrachten.

Die Auslagerung der DDC ist aus mehreren Gründen erforderlich:

1. Die digitale Regelung ist eine zeitkritische Funktion. Sie muß in einem definierten Zeitraster (Abtastrate) unabhängig von der Gesamtsystembelastung die Regelalgorithmen abarbeiten.

2. Die Funktionssicherheit wird durch die Auslagerung auf einen eigenen Prozessor erhöht. Die DDC, die die Prozeßgrößen des Prüfstandes kontrolliert, ist auch dann noch funktionsfähig, wenn Störungen in anderen Teilen des Mehr-Prozessor-Systems auftreten.

3. Die Trennung der prozeßorientierten Funktionen von den ablauf- bzw. kommunikationsorientierten Funktionen unterstützt das modulare Konzept des Prüfstand-Steuersystems und erleichtert damit eine Adaption an zukünftige Anwendungsfälle.

11.2.1 PID-Regelalgorithmus

Für die Regelung der analogen Prozeßgrößen werden diskrete PID-Algorithmen eingesetzt. Die Systemanalyse hat ergeben, daß die für modernere, leistungsfähigere Algorithmen benötigten Regelstreckendaten /39/40/ bei der zu bedienenden Anlage nicht zur Verfügung stehen. Experimentell wurde nachgewiesen, daß die Leistungsfähigkeit des PID-Algorithmus die Anforderungen der vorliegenden Anwendung erfüllt /41/.

Das Verhalten eines kontinuierlichen Reglers kann in den diskreten Bereich übertragen werden, wenn man von der Beschreibung im Zeitbereich ausgeht,

$$u(t) = k_c \left\{ e(t) + \frac{1}{T_i} \int_0^t e(t)\frac{dt}{t} + T_d \frac{d}{dt} e(t) \right\}$$

wobei

$$e(t) = r(t) - c(t)$$

der auftretende Fehler am Reglereingang ist. Bei r(t) handelt es sich um die Führungsgröße, c(t) ist die Regelgröße und u(t) die Reglerausgangsgröße (Stellgröße).

Nach /36/ ergibt sich hieraus die diskretisierte Form

$$u_k = u_{k-1} + \Delta u_k$$

$$= u_{k-1} + K_p(c_{k-1}-c_k) + K_1(r_k-c_k) +$$

$$K_D(2c_{k-1}-c_{k-2}-c_k),$$

wobei Δu_K die Differenz zwischen der k-ten und (k-1)-ten Stellgröße ist.

11.2.2 Aufbau der Funktionseinheit DDC

Die Funktionseinheit DDC wird durch einen Verbund von fünf Doppel-Europakarten realisiert (vgl. Abb. 8.2.1-1):

- DDC-Prozessor-Karte (DDC)
- Intelligente Analog-Eingabe-Karte (IAE)
- Intelligente Digital-Eingabe-Karte (IDE)
- 2 x Analog-Ausgabe-Karte (AA).

Die DDC-Rechner-Karte ist ein Standardprodukt aus dem AMS-Mikrorechner-Kartensystem der Firma Siemens. Sie ist als Bus-Master ausgelegt und hat in der Prüfstand-Steuereinheit die höchste Priorität. Die intelligente Analog-Eingabe-Karte (IAE) ist ein Bus-Slave. Da sie selbst den Systembus nicht bedienen kann, kommuniziert die Masterkarte mit ihr über ihren Dual-Port-Speicher. Die Intelligente Digital-Eingabe-Karte (IDE), ein Bus-Master, schreibt digitalisiert vorliegende Meßwerte in den gemeinsamen Speicher, wo sie der DDC-Karte zugänglich sind. Die Analog-Ausgabe-Karten, Slaves ohne eigene Intelligenz, werden ebenfalls von dem Master (DDC) bedient.

11.2.3 Programmsystemaufbau für das DDC-Modul

Die DDC-Masterkarte läuft, wie die übrigen Bus-Master, unter dem implementierten Betriebssystem.

Neben der Betriebssystemsoftware verfügt das Modul über zwei Anwendertasks (vgl. Abb. 11.2.3-1):
INIT für die Initialisierung und RGLUNG, das eigentliche Regelungsprogramm.
INIT führt nach einem Systemrestart einen Selbsttest des DDC-Moduls durch. Dieser umfaßt:

- EPROM-Test mit Checksum Prüfung, Funktionstest des Privat-RAMs der DDC-Prozessorkarte,
- Funktionstest des Timerbausteins und des Interruptcontrollers.

Nach Abschluß des Selbsttests wartet INIT auf die "Fertig"-Meldung des Selbsttestmoduls STM der intelligenten Analog-Eingabe (IAE), die ihrerseits ebenfalls einen Selbsttest durchgeführt hat. Wurde während des Selbsttests ein Fehler entdeckt, wird dies über ein Error-Flag dem Ablaufmodul (ABL) gemeldet, das darauf das System sperrt und an den Bediener eine Fehlermeldung ausgibt. Nach erfolgreichem Selbsttest aller DDC-Komponenten werden die internen Tabellen des DDC-Moduls initialisiert. Dies umfaßt die Übergabe von Steuerinformationen von der DDC-Prozessorkarte an das Programm-Modul SRV der IAE, die Übertragung der Regelkreisparameter aus dem gemeinsamen Speicher in die Datenblöcke der einzelnen Regelkreise und Rücksetzen aller Stellgrößen. Danach wird eine "Fertig"-Meldung an die Ablaufsteuerung geschickt und die Kontrolle an die DDC-Task RGLUNG übergeben. Die Initialisierungstask INIT hat damit die Aufgabe erfüllt und suspendiert sich selbst.

Die Regeltask RGLUNG wird im 100 msec-Takt zyklisch durchlaufen und bedient die 12 Regelkreise im Zeitmultiplexverfahren. Der Zeittakt (Interrupt) wird mit einem programmierbaren Taktgenerator erzeugt.

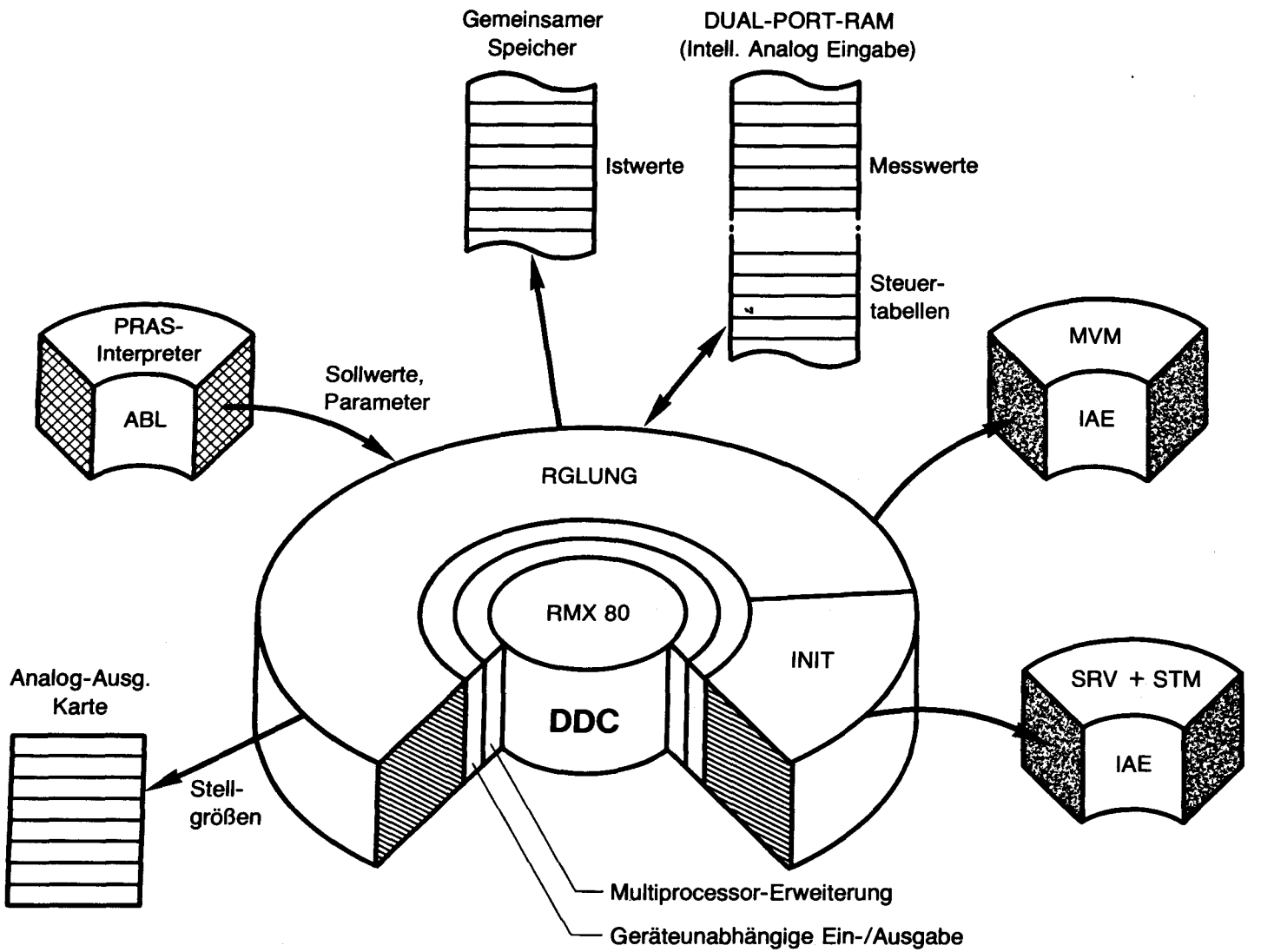


Abb. 11.2.3-1 Programmsystem Digitale Regelung

Nach dem Start durch den Zeittakt wird das Regelprogramm 12mal (für jeden der 12 Regelkreise) durchlaufen. Bei jedem Durchlauf wird

- der Istwert eingelesen,
- der PID-Algorithmus abgearbeitet,
- die Stellgröße ausgegeben,
- die internen Tabellen aktualisiert,
- der Istwert in den gemeinsamen Speicher geschrieben.

Bei der Programmentwicklung mußte beachtet werden, daß das Regelprogramm auch im Worst Case zwischen zwei Zeittakten, d.h. in weniger als 100 msec abgearbeitet werden kann, da ein Verlust der Synchronisation zu schlechten Regelergebnissen führt. In Abb. 11.2.3-2 ist der Programmablauf der Task RGLUNG in einem Flußdiagramm dargestellt.

11.2.4 Steuerblöcke des DDC-Moduls

Zur Verwaltung der Regelkreise werden Steuerblöcke (structured lists) im gemeinsamen Speicher und im Privat-RAM des DDC-Moduls benutzt.

Im gemeinsamen Speicher stehen die Sollwerte im Rumpfdatenblock RUMPDB. Dieser enthält für jeden der 12 Regelkreise

1 Byte Zustandsanzeige (Regelkreis aktiv/suspendiert)
4 Bytes Sollwert (Gleitkomma-Zahl)

Diese Werte werden von der Task PRAS der Ablaufsteuerung in den Steuerblock eingetragen und von der Task RGLUNG des DDC-Moduls ausgelesen.

Der Datenblock ISTWCM steht ebenfalls im gemeinsamen Speicher und enthält die Istwerte der 12 Regelkreise sowie die fünf weiterer analoger Prozeßgrößen, die von dem Steuerungssystem erfaßt, jedoch nicht beeinfluß werden können. Die Werte werden vom DDC-Modul als 4-Byte-Gleitkommazahlen

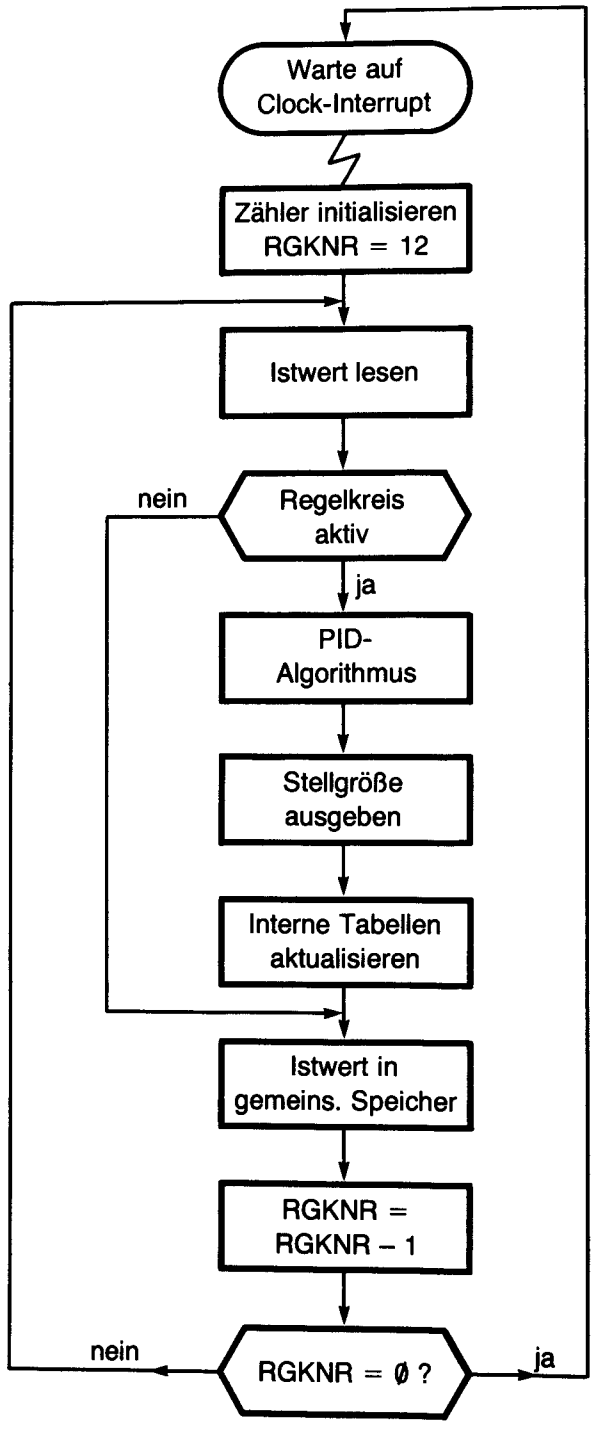


Abb. 11.2.3-2 Flußdiagramm der Task RGLUNG

eingetragen und nach jedem Abtastzyklus der DDC aktualisiert. Die Parameter der Regelkreise sind in der Liste PARLIS eingetragen. Für jeden Regelkreis sind abgespeichert:

1 Byte	zur Angabe des Reglertyps (P-, PI- oder PID-Regler)
1 Byte	Abtastrate (als Vielfaches von 100 msec)
4 Bytes	Parameter des Proportionalanteils KP (4-Bytes-Gleitkommazahl)
4 Bytes	Parameter des Integralanteils KI
4 Bytes	Parameter des Differentialanteils KD.

Die Regelkreisparameter werden bei Systeminitialisierung von der Ablaufsteuerung in den gemeinsamen Speicher eingetragen und können mit Hilfe eines Diagnose- und Testmoduls on-line modifiziert werden. Hierdurch hat der Prüfingenieur die Möglichkeit, das Regelkreisverhalten während des Betriebes der Anlage zu optimieren.

LÄNGE	NAME	FUNKTION
1 Byte	RGKACT	Zustandsflag
4 Byte	RK	Sollwert
1 Byte	REGTY	Reglertyp: 0=P, 1=PI, 2=PID
1 Byte	SMPLRT	Samplerate (Ganzz. vielf. von 0,1 sec.)
4 Byte	KP	Regelparameter KP (proportional)
4 Byte	KI	Regelparameter KI (integral)
4 Byte	KD	Regelparameter KD (differential)
1 Byte	TAKT	Takt
4 Byte	CK	Istwert der aktuellen Abtastperiode
4 Byte	CKM1	Istwert der vorigen Abtastperiode
4 Byte	CKM2	Istwert der vorvorigen Abtastperiode
2 Byte	STELSS	Stellgröße der aktuellen Abtastperiode
4 Byte	STSFM1	Stellgröße der vorigen Abtastperiode

Tabelle 11.2.4-1 Aufbau der Regelkreis-Steuerblöcke

Im Privat-RAM des DDC-Prozessors liegen Steuerblöcke für jeden der 12 Regelkreise. Sie haben eine Länge von 38 Bytes und enthalten alle Daten, die der PID-Algorithmus zur Bedienung der Regelkreise benötigt (siehe Abb. 11.2.4-1).

11.2.5 Integration der Digitalen Regelung in die Prüfstand-Steuereinheit

Das DDC-Modul kommuniziert mit den übrigen Teilnehmern am Multiprozessor-Bus über das in Kap. 10.1 beschriebene Kommunikationsverfahren.

Es übernimmt von der Ablaufsteuerung (ABL) die Sollwerte und Regelparameter der 12 Regelkreise in dem Netto-Datenbereich einer RMX80-Botschaft und übergibt die Istwerte in einen reservierten Datenblock im gemeinsamen Speicherbereich (AMS 128-A2 Mailbox), wo sie allen Busteilnehmern zur Verfügung stehen. Abb. 11.2.3-1 zeigt die Kommunikationspfade zwischen der DDC-Prozessorkarte und den übrigen Busteilnehmern.

Als Beispiel für die Kommunikation unter Nutzung der Funktionen des Multi-Prozessorbetriebssystems wird die Übergabe eines Sollwertes von der Ablaufsteuerung an das DDC-Modul in Abb. 11.2.5-1 detailliert beschrieben.

Die Task PRAS der Ablaufsteuerung schreibt den Sollwert und das Zustandsflag in die Tabelle RUMPDB (1) und sendet danach eine Botschaft, die einen Pointer auf die Sollwert-Tabelle enthält, an den Exchange DDCRQX (2). Diese Botschaft wird von der Task RGLUNG des DDC-Moduls empfangen (3) und der Sollwert mit Hilfe des Pointers aus der Tabelle RUMPDB in einen Regelkreis-Steuerblock transferiert (4). Als Bestätigung für den Empfang des Sollwertes schickt RGLUNG eine Quittungsbotschaft (5) über den Exchange DDCRSX an die Ablaufsteuerung zurück (6).

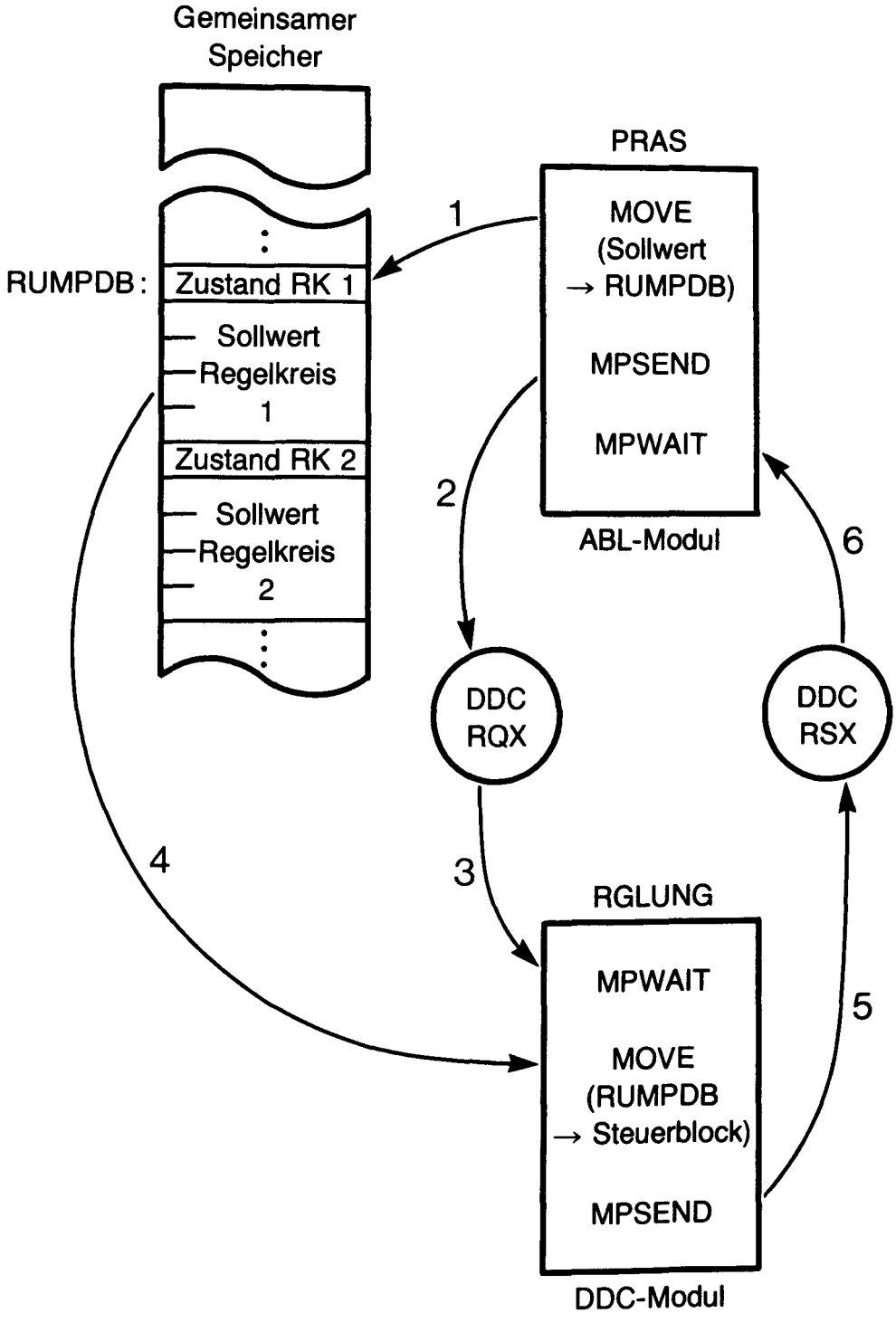


Abb. 11.2.5-1 Übergabe der Sollwerte von der Ablaufsteuerung an die Digitale Regelung (DDC)

Der Datenaustausch mit dem Meßwert-Verarbeitungsmodul MVM der intelligenten Analog-Eingabekarte (IAE) verläuft über das Dual-Port-RAM der IAE mit Hilfe einer einfachen Übergabeprozedur. Dabei fordert die Task RGLUNG die IAE mit einem Interrupt auf, aktuelle Meßwerte im Dual-Port-RAM bereitzustellen und den Zugriff auf das Dual-Port-RAM freizugeben. Mit einem Flag signalisiert RGLUNG der IAE die Beendigung des Datentransfers.

Die Ausgabe der Stellgrößen an die Prozeßperipherie erfolgt durch Beschreiben eines Ausgaberegisters auf der Analog-Ausgabekarte (AA).

11.2.6 Programmaufbau Intelligente Analog Eingabe

Das intelligente Analog-Eingabe-Modul dient der autarken Aufbereitung von Prozeßgrößen. Es besitzt folgende Teilfunktionen:

- Erfassen der analogen Prozeßmeßwerte
- Skalierung der Meßwerte
- Digitale Glättung (gleitende Mittelwertbildung)
- Linearisierung von Meßwertaufnehmerkennlinien und Offsetkompensation

Im Gegensatz zu den anderen intelligenten Busteilnehmern laufen die Programme in einem vorgegebenen Zyklus ab; deshalb wurde auf den Einsatz eines Betriebssystems verzichtet. Dies führt darüber hinaus zu einer signifikanten Beschleunigung des Programmablaufs, was eine schritthaltende Bereitstellung der Meßwerte für den DDC-Algorithmus auch bei umfangreicher Meßwertvorverarbeitung ermöglicht.

Alle Operationen laufen unter Kontrolle von Steuerblöcken ab, welche bei der Beschreibung der einzelnen Teilmodule näher erläutert werden.

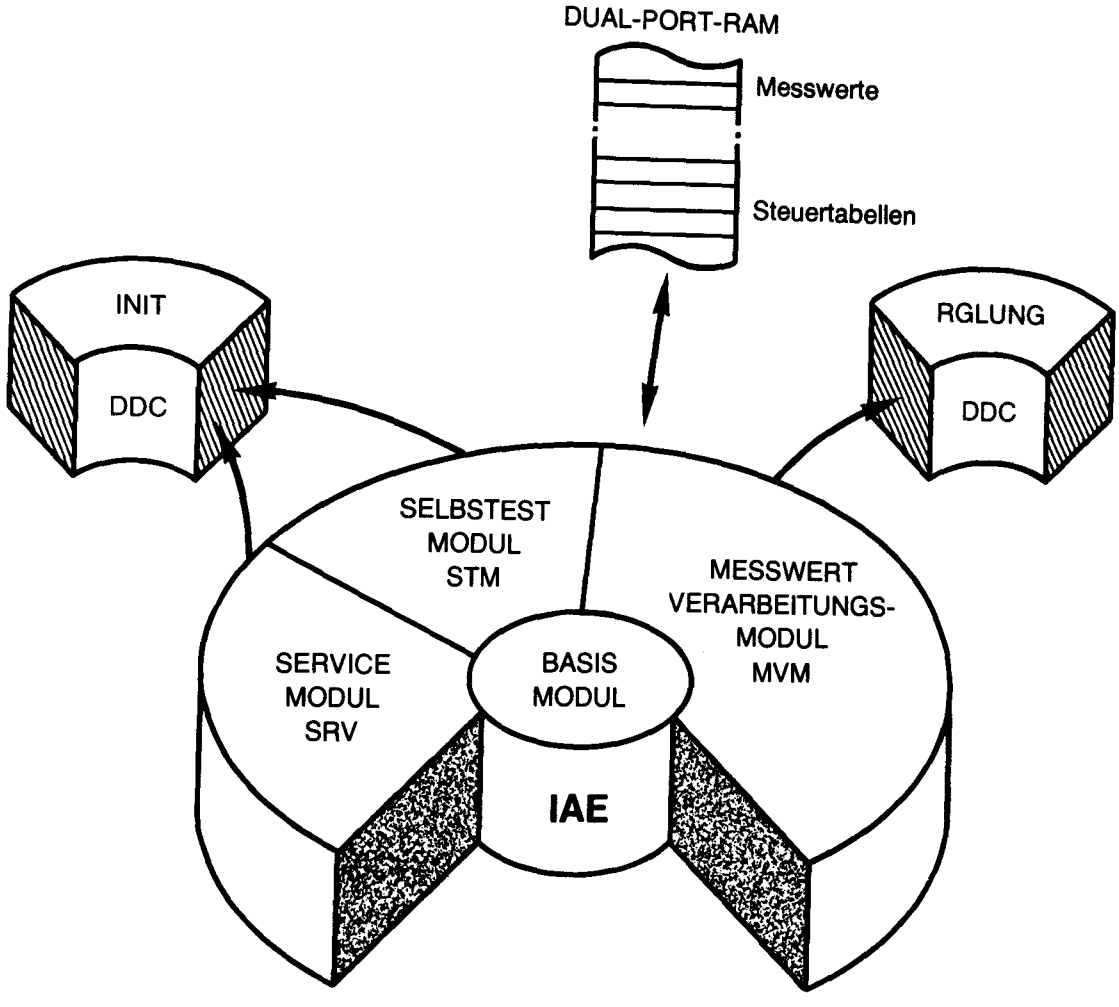


Abb. 11.2.6-1 Programmsystem Intelligente Analog Eingabe

Zur Realisierung dieser Funktionen wurden folgende Module implementiert:

- Basis-Modul
- Selbsttest-Modul (STM)
- Meßwertverarbeitungs-Modul (MVM)
- Service-Modul (SRV)

11.2.6.1 Programm-Module

- Basismodul

  Das Basismodul initialisiert die Hardware und startet den Selbsttest und die Meßwertverarbeitung.

- Selbsttestmodul

  Nach Einschalten der Versorgungsspannung bzw. Reset führt das IAE-Modul in Verbindung mit dem DDC-Prozessor einen Selbsttest durch. Nachstehende Abbildung zeigt Umfang und Ablauf des Tests.

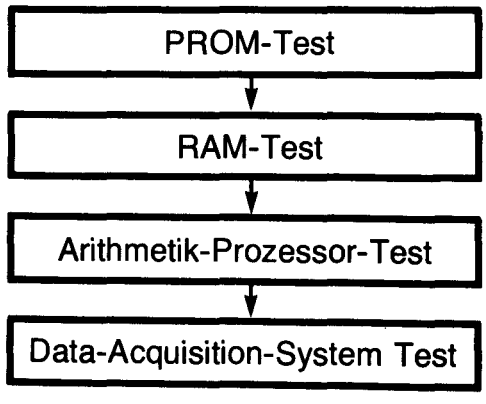


Abb. 11.2.6.1-1 Einzelfunktionen der Selbsttestroutine

Nach dem Test übergibt das IAE-Modul dem DDC-Prozessor das Testergebnis über das Dual-Port RAM.

- Meßwertverarbeitungsmodul

  Abb. 12.2.6.1-2 zeigt den Ablauf der Meßwertverarbeitung

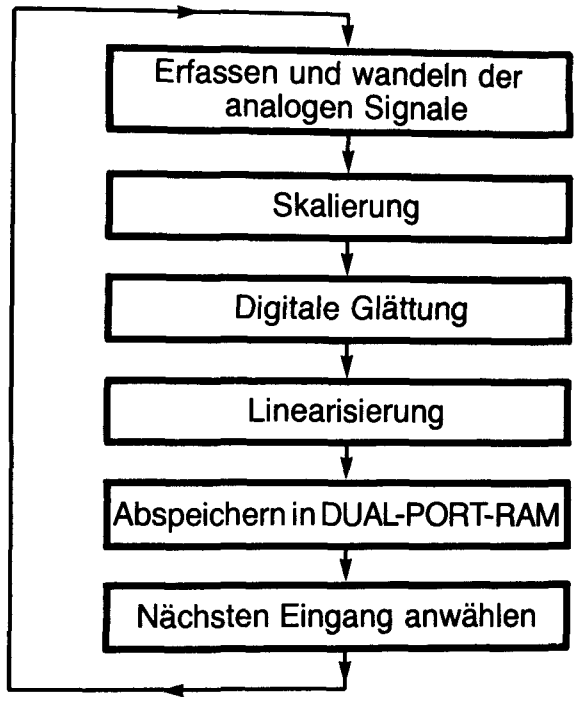


Abb. 11.2.6.1-2 Ablauf der Meßwertverarbeitung

Bei der Erfassung der analogen Eingangssignale greift das Modul auf eine Steuertabelle zu, in der die Zuordnung zwischen Meßgröße und physikalischer Adresse eingetragen ist. Der nach der A/D-Wandlung vorliegende 12-Bit Wert wird in ein 4 Byte Floating-Point-Format umgewandelt und einer Skalierungsrechnung unterzogen.

Die digitale Glättung (gleitende Mittelwertbildung) wird in der Form vorgenommen, daß eine steuertabellenabhängige Anzahl n von Einzelwerten der n letzten Messungen der analogen Prozeßgrößen in einem Ringpuffer gespeichert und zur Mittelwertbildung herangezogen wird. Die Einzelwerte im Ringpuffer werden zyklisch erneuert, wobei der älteste Meßwert durch den aktuellen ersetzt wird.

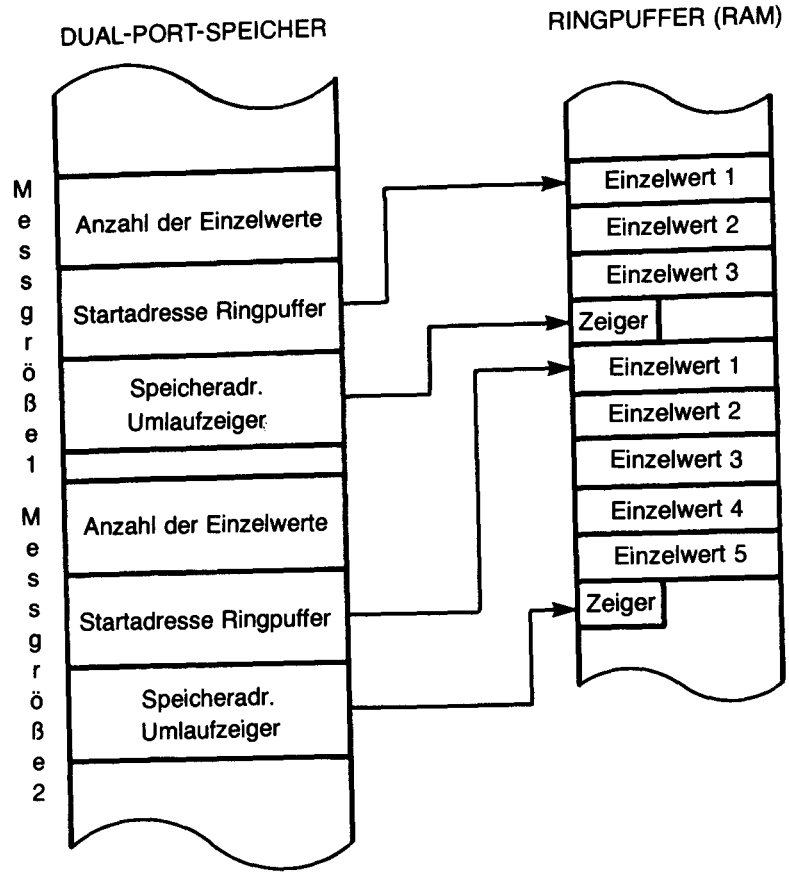


Abb. 11.2.6.1-3 Steuertabelle für digitale Glättung und Pufferbelegung

Die Steuertabelle enthält für jede Meßgröße folgende Informationen:

- Die Anzahl der aufeinanderfolgenden Einzelmeßwerte, welche zur Mittelwertbildung herangezogen werden
- Die jeweilige Startadresse des zugehörigen Ringpuffers
- Die Speicheradresse des Umlaufzeigers

Mit dem Linearisierungsmodul werden Linearitäts- und Offsetabweichungen der Meßwertaufnehmer (Transmitter) korrigiert. Dazu wird die Transmitterkennlinie in eine ausreichend große Anzahl von linearen Teilstücken (Geraden) zerlegt.

- Servicemodul

  Das Servicemodul stellt eine Kalibrierhilfe für das IAE-Modul dar. In Verbindung mit dem DDC-Prozessor können spezielle zum Abgleich vorbereitete Eingangskanäle auf einem Display dargestellt und zum Abgleich des Data-Acquisition-Systems herangezogen werden.

11.2.7 Programmaufbau Intelligente Digitale Eingabe (IDE)

Das IDE-Modul ist auf einem AMS-D2-Board implementiert und dient zur Erfassung von Meßwerten, die die Prozeßperipherie in digitalisierter Form liefert.

Auf der IDE sind zwei Programmodule, INIT und MWED, implementiert.

Mit dem Modul INIT wird die Initialisierung der boardeigenen Hardware vorgenommen und ein Selbsttest durchgeführt.

Das Modul MWED steuert die Erfassung der digitalen Meßwerte. Es handelt sich dabei um eine Impulskette drehzahlproportionaler Frequenz und um BCD-codierte Durchfluß- und Temperaturwerte. Nach der Erfassung werden diese Werte umgewandelt und im gemeinsamen Speicher zur Weiterverarbeitung abgelegt.

11.3 Bedienfeldsteuerung (BDF)

Die Bedienfeldsteuerung wurde auf Basis der Mikrorechnerkarte AMS-D3 in Verbindung mit der in Kapitel 8.2.1.1 beschriebenen Speichererweiterung implementiert. Das Modul stellt zusammen mit Farbsichtgerät, Funktions- und Zifferntastatur die Schnittstelle zum Prüfstandbediener her. Die in Kapitel 6 kurz skizzierten Funktionen der Bedienfeldsteuerung werden im folgenden ausführlicher beschrieben.

11.3.1 Programmaufbau der Bedienfeldsteuerung

Neben der Systemsoftware mit RMX80-Kern, Multiprozessorerweiterung und geräteunabhängiger Ein/Ausgabe laufen auf der Bedienfeldsteuerung die folgenden Anwendertasks:

- INIT zur Systeminitialisierung
- PREAD Verarbeitung der Bedienereingaben
- PWRITE Ausgabetask für Farbmonitor
- IMSECO Funktionsorientierte Bedienerführung
- PRASIO Ein-/Ausgabefunktion für Ablaufprozessor
- DPYANA Echtzeitdarstellung der Prozeßgrößen

Das Programmsystem mit den wesentlichen Kommunikationspfaden ist in Abb. 11.3.1-1 wiedergegeben.

11.3.1.1 Initialisierung

Nach Einschalten der Stromversorgung bzw. Reset führt die Task INIT einen Selbsttest der boardeigenen Hardware durch. Anschließend werden für die Bedienerführung notwendigen Bildmasken vom Massenspeicher des Zentralsystems in den Arbeitsspeicher der BDF geladen. Dazu wird die geräteunabhängige Ein/Ausgabe des Multiprozessor-Betriebssystems benutzt. Nach Abschluß der Initialisierung wird das System für den Bediener und für die Ablaufsteuerung freigegeben.

11.3.1.2 Verarbeitung der Bedienereingaben

Die Verarbeitung der Bedienereingaben wird durch die Task PREAD vorgenommen. Dabei werden zwei Eingaben unterschieden:

- vom Bediener initiierte Eingabe über die Funktions-/Zifferntastatur
- programmgesteuerte Eingabe über die Tasks IMSECO und PRASIO

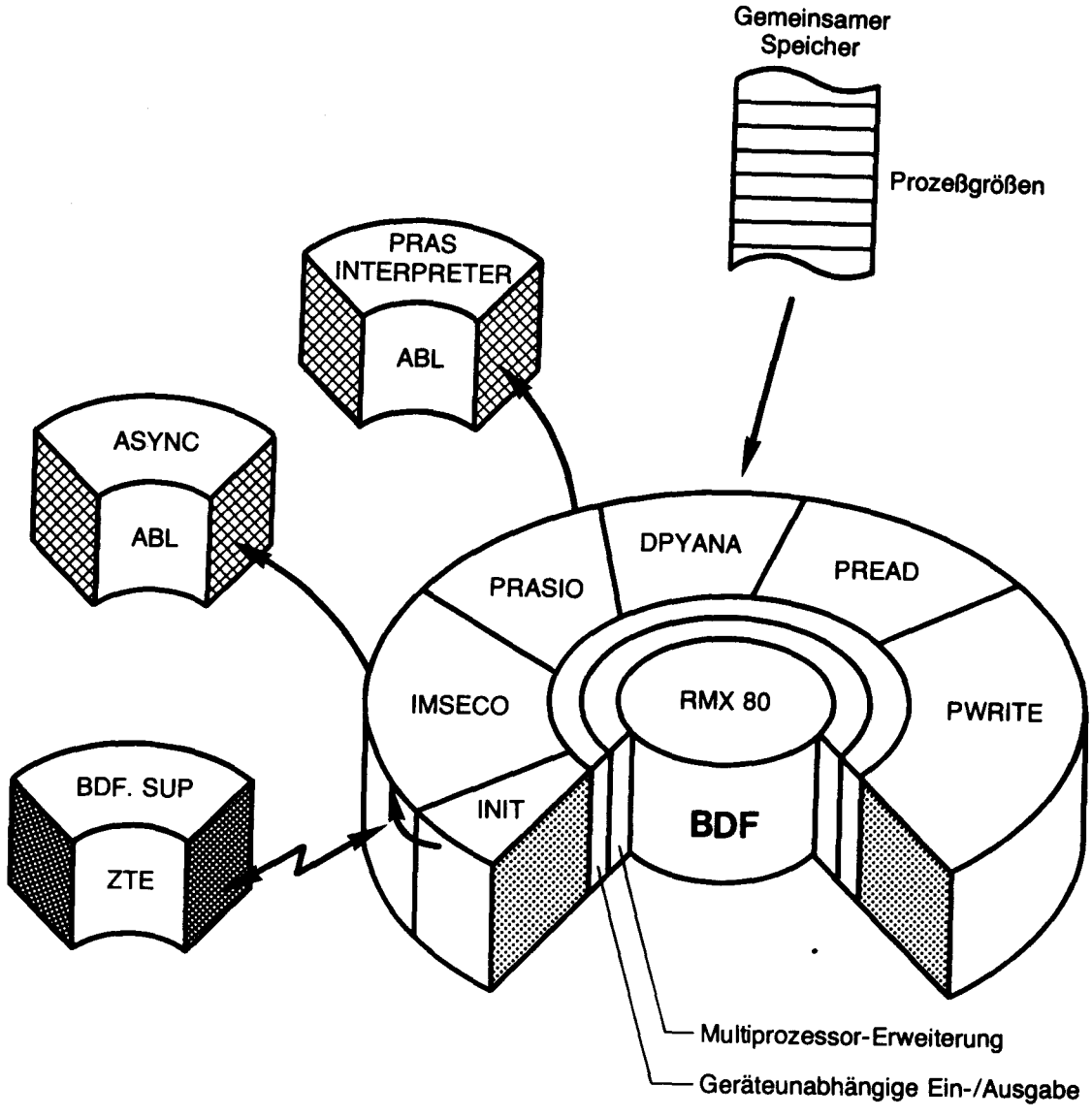


Abb. 11.3.1-1 Programmsystem Bedienfeldsteuerung

Bei der Tastatureingabe wird ein der angewählten Funktion entsprechender Code an die Task IMSECO übergeben, die die weitere Funktionsabarbeitung übernimmt.

Bei der programmgesteuerten Eingabe (z. B. Meßwerteingabe) wird der Task PREAD eine Eingabesteuertabelle übergeben mit Informationen für den Farbmonitor (Cursorposition Vordergrund/Hintergrund) sowie die Eingabefeldlänge und ein Funktionscode. Über diesen wird Art und Umfang der Plausibilitätsprüfung bestimmt. Nach Abarbeitung der Eingabesteuertabelle sendet PREAD die eingegebene Information an die anfordernde Task.

11.3.1.3 Ausgabe auf Farbmonitor

Die Ausgaben auf dem Farbmonitor werden über die Ausgabetask PWRITE abgewickelt. Dazu werden dieser Task die Nummer einer Ausgabesteuertabelle sowie der Zeiger zu einer Bildmaske bzw. einem Ausgabepuffer übergeben. Die Ausgabesteuertabelle enthält die Steuerinformationen für den Farbmonitor und die Anzahl der auszugebenden ASCII-Zeichen, während der Zeiger die Adresse der auszugebenden ASCII-Zeichen enthält.

11.3.2 Graphische Darstellung der Prozeßgrößen

Die Task DPYANA erzeugt in Echtzeit eine graphische Darstellung der Prozeßgrößen auf dem Farbmonitor. Die Meßwerte werden numerisch und quasi-analog (als Histogramm) angezeigt. Die Schaltzustände der binären Stellgrößen werden durch farbliche Unterlegung von Schriftfeldern eingeblendet. Die Prozeßgrößendarstellung kann vom Bediener über die Task IMSECO und vom Prüfprogramm über PRASIO aktiviert werden. Die Darstellung nach Abb. 11.1.1.3-2 aufgebaut, wobei die Eingabespalte für die Sollwerte ausgeblendet wird.

11.3.3 Ein/Ausgabefunktion für Ablaufprozessor

Die Task PRASIO stellt ihre Ein/Ausgabe-Dienste dem PRAS-Interpreter als ausgelagerte Funktionen zur Verfügung. Der Interpreter ruft über Steuerzeichen (vgl. Kapitel 12.7 bis 12.9) vorformatierte Bildmasken und Eingabeformulare auf.

11.3.4 Funktionsorientierte Bedienerführung

Die Bedienerführung wird von der Task IMSECO vorgenommen. In der Grundstellung wartet IMSECO auf eine Botschaft der Task PREAD. Nach Empfang ruft IMSECO das dem Funktionscode entsprechende Funktionsmodul auf und geht anschließend in

die Grundstellung zurück. Bei der Ausführung von Funktionen nimmt IMSECO Verbindung zum Zentralsystem (Task BDF.SUP) und zum Ablaufmodul (Task ASYNC) auf. Dazu werden die geräteunabhängige Ein/Ausgabe und die Multiprozessorerweiterungen des Betriebssystems benutzt. Die im folgenden beschriebenen Funktionen der Bedienerführung werden über die Ziffern/Funktionstastatur initiiert:

- FR400 Kurzbeschreibung der Einzelfunktionen (Menü)

- Organisatorische Funktionen

  - SPA Start Prüfablauf
  - UPA Unterbrechung Prüfablauf
  - EPA Ende Prüfablauf

- Betriebsarten

  - SEA Semi-Automatik-Betrieb
  - SBY Stand-By-Betrieb

- Display-Funktionen

  - DPY Display Prüfdaten

- Funktionsmenü FR400
  Nach Eingabe der Programm-Nr. 400 erscheint auf dem Farbmonitor eine Auflistung und Kurzbeschreibung der Bedienfunktionen. Die Bezeichnung FR400 wurde zur Vereinheitlichung mit bereits in Betrieb befindlichen Automatisierungskomplexen gewählt (vgl. auch Kap.13).

- Start Prüfablauf SPA
  Bei Auslösen der Funktion SPA wird auf dem Farbmonitor die zugehörige Bildmaske mit den Konstant-Daten des Kraftstoffreglerkennsatzes ausgegeben. Nach Eingabe der Seriennummer durch den Bediener werden die Variablen-Daten des Kennsatzes vom Zentralsystem geladen und in die Maske eingeblendet. Im Zentral-

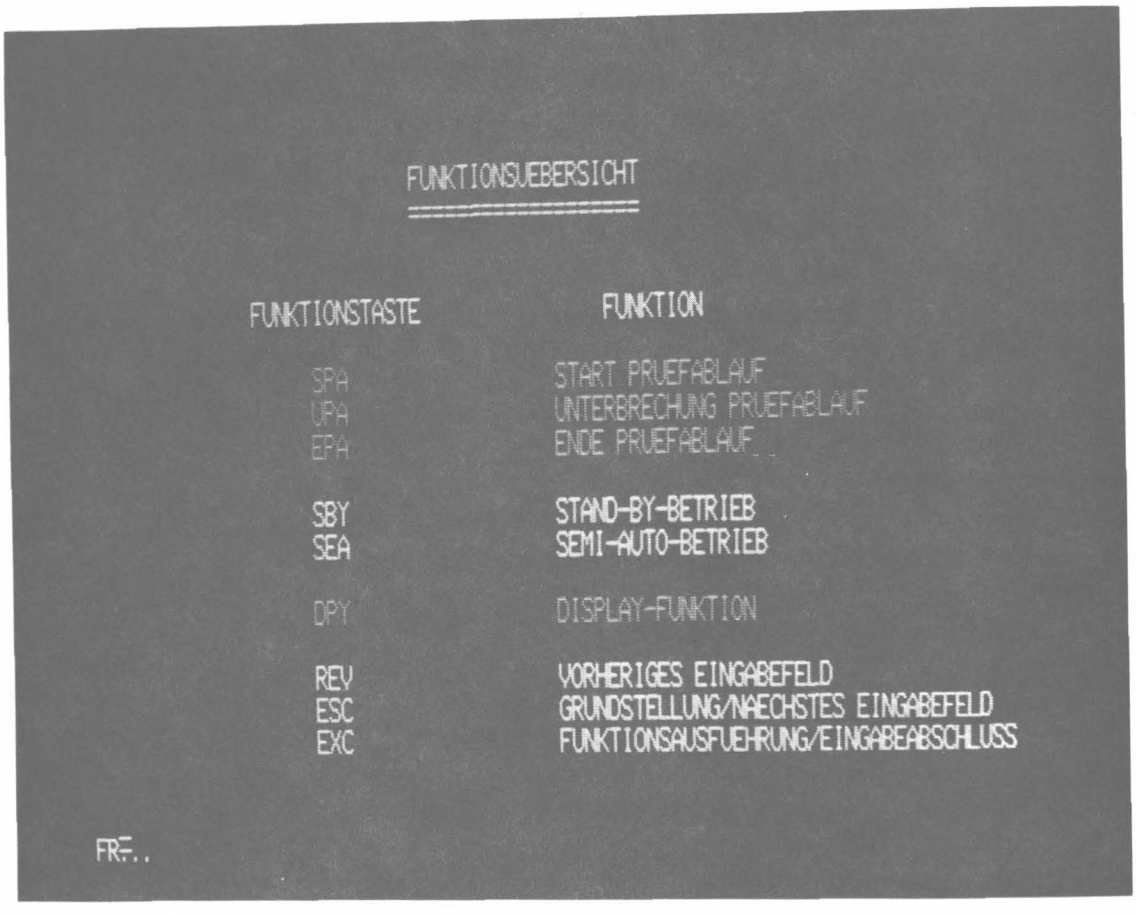


Abb. 11.3.4-1 Menü der Bedienfunktionen

system wird die zur Serien-Nummer gehörende Prüfdatei eröffnet und eine entsprechende Ausgabe auf dem Protokolldrucker durchgeführt.

Nach Ausgabe der Kennsatzvariablen werden vom Bediener folgende Eingaben erwartet:

- Personalnummer
- Prüfschrittnummer
- Einzelschrittcode

Mit dem Einzelschrittcode legt der Bediener fest, ob der Prüfablauf nach Abarbeiten des Prüfschritts beendet oder das Programm des nächstfolgenden Prüfschritts automatisch geladen und gestartet werden

soll. Das Laden des Prüfschritts wird vom Ablaufsteuermodul durchgeführt. Dazu wird der Task ASYNC auf dem ABL die Prüfschrittnummer und der Einzelschrittcode übergeben.

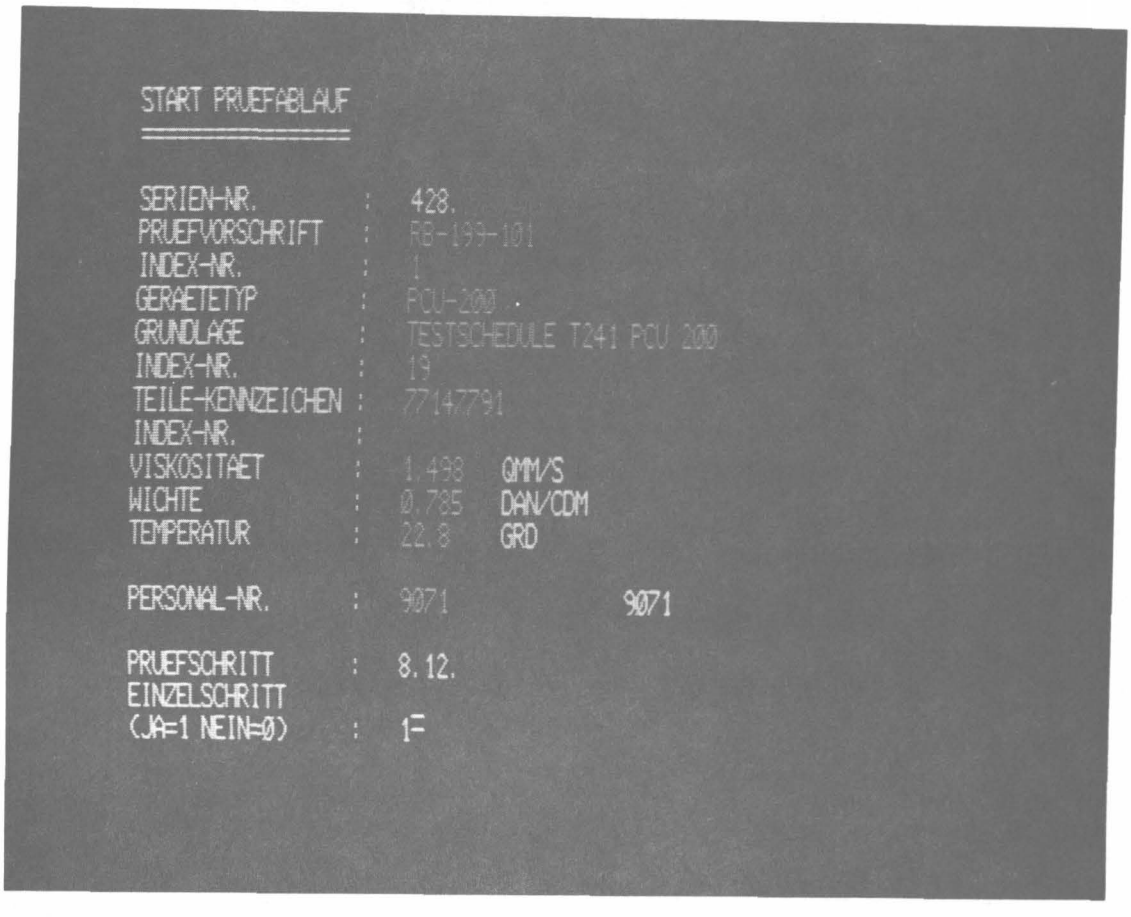


Abb. 11.3.4-2 Bildmaske "START PRÜFABLAUF" mit organisatorischen Daten

- Unterbrechung Prüfablauf UPA
  Mit der Funktion UPA wird der Prüfablauf unterbrochen. IMSECO sendet dem Zentralrechner und dem Ablaufsteuermodul einen Unterbrechungscode. Der Zentralrechner schließt darauf die Prüfdatei und gibt eine entsprechende Meldung auf dem Protokolldrucker aus. Die ABL fährt alle Prozeßgrößen in einen sicheren Zustand. Auf dem Farbmonitor erscheint die Bildmaske "Unterbrechung" mit Seriennummer- und Prüfschrittangabe.

- Ende Prüfablauf EPA
  Die Funktion EPA entspricht im wesentlichen der Funktion UPA. Zusätzlich wird im Zentralsystem überprüft, ob alle Prüfdateien vollständig sind. Ist dies nicht der Fall, wird ein Fehlercode an IMSECO zurückgegeben. Auf dem Monitor erscheint die Maske "Ende Prüfablauf".

- Semi-Automatik-Betrieb SEA
  Mit der Funktion SEA ist ein teilautarker Betrieb der Prüfstandsteuerung ohne das Zentralsystem möglich. Auf dem Farbmonitor wird ein Submenü ausgegeben, aus dem der Bediener die benötigte Teilfunktion auswählen kann. Eine detaillierte Beschreibung und Abbildung dieser Funktionen befindet sich in Kapitel 11.1.1.3.

- Stand-By-Betrieb SBY
  Mit der SBY-Funktion wird der Prüfstand ähnlich wie bei der Funktion "Unterbrechung" in einen inaktiven Zustand gebracht. Die vorher eingestellten Konditionen werden jedoch gespeichert. Auf dem Monitor wird die SBY-Maske ausgegeben und eine Reaktivierungs-Eingabe erwartet. Nach erfolgter Eingabe wird der Prüfstand in den vorherigen Betriebszustand gebracht und der Prüfablauf fortgesetzt.

- Display-Funktion DPY
  Die Display-Funktion besteht aus zwei Teilfunktionen:

  a) Display - Echtzeitdarstellung der Prozeßgrößen

     Hier wird die Task DPYANA aktiviert, welche den Bildaufbau zur Prozeßgrößendarstellung (vgl. 11.3.5) vornimmt.

  b) Display - Prüfdatei

     Mit dieser Funktion lassen sich Prüfdaten vom Massenspeicher laden und auf dem Farbmonitor darstellen. Der Bediener gibt Serien- und Prüf-

schrittnummer ein. Die Task IMSECO lädt die Prüfdatei und generiert auf dem Monitor aus den RAM-residenten Konstantanteilen (Textzeilen, Einstellwerte, Limitwerte) und den Prüfdaten das Ausgabebild. Umfangreichere Prüfdateien werden in aufeinanderfolgenden Teilbildern dargestellt.

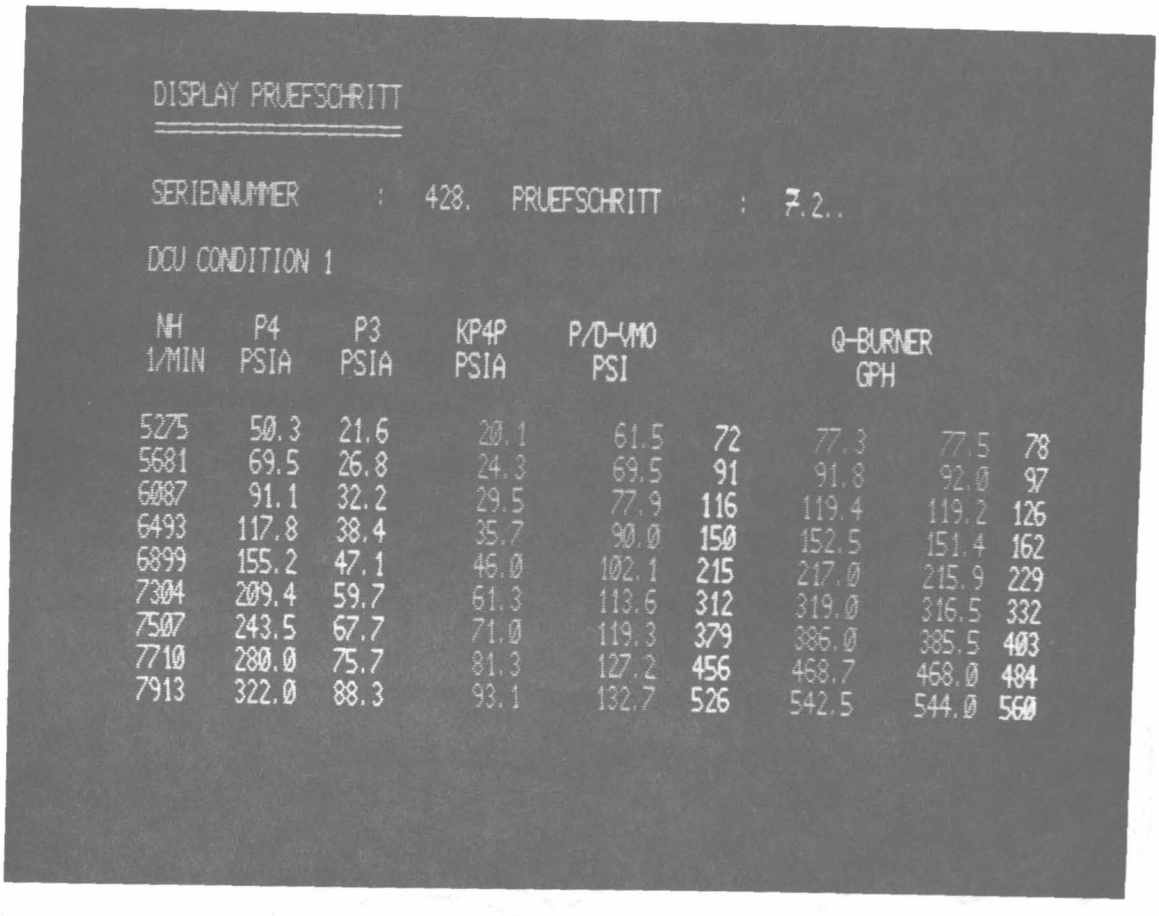


Abb. 11.3.4-3 Darstellung einer Prüfdatei auf dem Bedienfeldmonitor

11.4 Zentralrechnersystem (ZTE)

Die Aufgaben dieser Funktionseinheit wurden im Kapitel 6 bereits aufgezeigt und werden im wesentlichen von 7 Systemprogrammen wahrgenommen. Die zunächst vorgesehene Meßdatenverarbeitung soll zu einem späteren Zeitpunkt auf einem übergeordneten Betriebsrechner vorgenommen werden. Zur Unterstützung eines reibungslosen Betriebsablaufs wurden zusätzlich einige Hilfs- und Testprogramme implementiert.

Der Aufbau der Funktionseinheit ist in Abb. 11.4.1-1 wiedergegeben. Die problemorientierten Dienstprogramme laufen ohne Ausnahme auf dem Einkarten-Mikrorechner (iSBC80/30). Die in Abb. 8.1-1 dargestellten weiteren passiven, intelligenten Module dienen ausschließlich der Unterstützung der in Kap. 10 beschriebenen Betriebssystemfunktionen.

11.4.1 Programmsystem

Nach Abb. 11.4.1-1 bauen sich die funktionsbestimmenden Programme um das Betriebssystem mit RMX80-Kern, Multiprozessorerweiterung und geräteunabhängige Ein/Ausgabe auf. Im Gegensatz zu den in den Prüfstandssteuereinheiten beschriebenen Programmsystemen muß wegen des Umfangs der Programme ein Teil auf den Massenspeicher ausgelagert werden können.

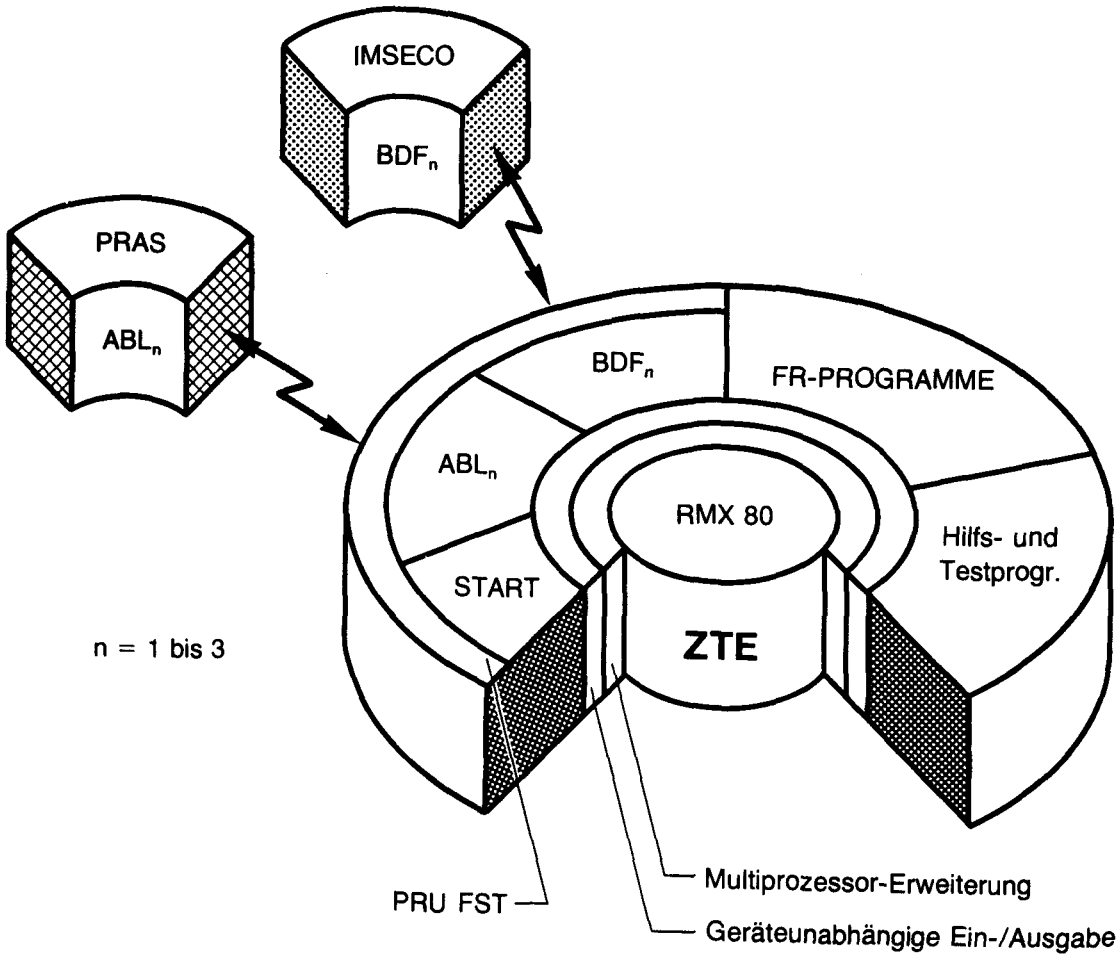


Abb. 11.4.1-1 Programmsystem Zentralrechner

- Programm PRUFST
  Dieses Programm bildet die Basis für einen automatisierten Prüfablauf. Durch Eingabe der Prüfstandnummern wird der Betrieb an den einzelnen Prüfständen ermöglicht und gleichzeitig auf dem Protokolldrucker die Kopfzeile des Betriebsprotokolls ausgedruckt (vgl. Abb. 11.4.1-2).

```
DATUM: 82/11/12
ZEIT   PRUEFSTAND 4P811        PRUEFSTAND 4P822        PRUEFSTAND 4P822
-------------------------------------------------------------------------
06:32  BEREIT                  BEREIT                  NICHT BEREIT
06:35  SN:809  **A
06:36  SN:809  PS4.1
06:40                          SN:841  **A
06:46  SN:809  PS4.2           SN:841  PS4.53
06:47                          SN:841  PS4.6
06:48  SN:809  PS4.3
06:51  SN:809  **U
06:54                          SN:841  PS8.11
07:02                          SN:841  PS8.12
07:03  SN:809 **A
07:06  SN:809 PS4:3            SN:841  PS6.2
```

Abb. 11.4.1-2 Beispiel eines Betriebsprotokolls mit aktueller Uhrzeit, Serien- und Prüfschrittnummern für 3 Prüfstände

Im einzelnen gliedert sich das Programm in drei Tasks:

- START Systeminitialisierung
- ABLAUFSTEUERUNG, unterstützt Laden der Prüfprogramme und Abspeichern der Prüfdaten
- BEDIENFELD, stellt Teilfunktionen für die Bedienfeldsteuerung bereit
- LOGGER, generiert das Betriebsprotokoll

- Programm FR400
  Das Programm dient der Erstellung von Kennsätzen, die im wesentlichen die charakteristischen organisatorischen wie technischen Daten für die zu prüfenden Kraftstoffregler enthalten. Mit der Erstellung des Kennsatzes wird ein Freiraum für die Meßdaten auf dem Massenspeicher bereitgestellt und die Seriennummer für

den automatischen Prüfablauf freigegeben (vgl. Kap. 11.3.4). Der Kennsatz wird per Dialog mit dem Bediener auf Basis eines Freiformulars (Musterkennsatz Maske) generiert.

PLU	PRUEFPROTOKOLL	Triebwerk	Bl.	Seite
			12	1
		Nr. 1	Ausg. 81/11/03	

Benennung: FLOW CONTROL - ENGINE FUEL		Serien-Nr: 0502
Geraetetyp:	Teile Nr: 77147791	Index:
D.I.S. PL: 3115	Issue: 37	Grundlage: TESTSCHEDULE
Index: 19	Prueffluessigkeit: AVTUR D.ENG.RD.2494	
Viskositaet: 1.498 mm^2/s, 0330 Grad		Wichte: 0.785 daN/dm^3, 21.0 Grad
Monteur:	Kontrolleur:	Datum:
Pruefstands-Nr: 4P811	GPS-Neuss:	Datum:

Abb. 11.4.1-3 Kennsatz mit organisatorischen und technischen Daten (Prüfparameter), Eingabefelder unterstrichen.

Ohne Eingabe des Kennsatzes ist jeder automatische Prüfablauf ausgeschlossen. Zur bedienerfreundlichen Änderung der Eingabemaske wurde das Programm tabellengetrieben aufgebaut.

- Programm FR 401
  Unabhängig von der Eingabe des Kennsatzes kann dieser mit Hilfe des Programmes FR 401 auf dem Graphikdrucker ausgedruckt werden (vgl. Abb. 8.2.1-1).

- Programm FR 403
  Zur nachträglichen Eingabe handprotokollierter Meßwerte und Korrektur wurde eine zusätzliche Eingabemöglichkeit in den Meßdatenfile vorgesehen. Nach Start mit der Identifikation der Serien- und Prüfschritt-

nummer wird die zugehörige Maske mit den im Prüfdatenfile bereits vorhandenen Prüfdaten auf dem Sichtgerät dargestellt. Eingabefelder werden unterstrichen, die Eingabe selbst wird auf formale Plausibilität geprüft.

- Programm FR 404
  Nach Eingabe der Serien- und Prüfschrittnummer wird das Ergebnis eines Prüfablaufs auf dem Protokolldrucker ausgegeben. Dabei können alle Prüfschritte oder eine ausgewählte Teilmenge protokolliert werden.

- Programm PREDIT
  Der Editor zur Erstellung und Korrektur von Prüfablaufprogrammen wird in Kap. 12 näher erläutert.

- Hilfs- und Testprogramme
  Zur Eingabe der Uhrzeit und des Datums, Kopieren von Disketten, Ausgabe von Inhaltsverzeichnissen und Ausgabe von Dateien auf Bildschirm oder Graphik-Drucker wurden einige Systemdienste implementiert, die auch einem mit den Begriffen der Datenverarbeitung unerfahrenen Prüfstandsoperateur die Bedienung des zentralen Rechnersystems ermöglicht.

  Zur Wartung und Instandhaltung wurden einige Testprogramme geschrieben, die u.a. den Test der Kommunikationsstrecken zu den Prüfstandssteuereinheiten nach Vorgabe durch den Bediener am Bildschirm ermöglichen.

PLU	PRUEFPROTOKOLL	Triebwerk	Bl.	Seite
			12	1
		Nr. 1	Ausg. 81/11/03	

Benennung: FLOW CONTROL - ENGINE FUEL — Serien-Nr: 0502

Geraetetyp: — Teile Nr: 77147791 — Index: —

D.I.S. PL: 3115 — Issue: 37 — Grundlage: TESTSCHEDULE

Index: 19 — Pruefffluessigkeit: AVTUR D.ENG.RD.2494

Viskositaet: 1.498 mm^2/s, 0330 Grad — Wichte: 0.785 daN/dm^3, 21.0 Grad

Monteur:	Kontrolleur:	Datum:
Pruefstands-Nr: 4P811	GPS-Neuss:	Datum:

Blatt: 2 von 12 — FLOW CONTROL - ENGINE FUEL — Serien-Nr: 0502

4.1 PRESSURE RAISING VALVE SETTING

DURCHFLUSS GPH	DIFFERENZDRUCK PSI			BRENNERDRUCK PSIG
30	295	298	315	0
500	—	283	—	251
900	—	86	—	808

4.2 MINIMUM FLOW SETTING

P4 PSIA	atm	P3 PSIA	atm

DREHZAHL i/MIN	DIFFERENZDRUCK PSI VMO			DURCHFLUSS GPH		
4736	47	48	49	38.0	41.1	42.0
6748	—	92	—	54.3	57.7	60.3

4.3 VMO ECCENTRIC (LEVER) STOP SETTING, SLOPE SETTING AND PROFILE CHECK

DREH-ZAHL i/MIN	KP4P PSIA	DIFFERENZ-DRUCK PSI VMO	DURCHFLUSS GPH			
7710	20	105	163.0	163.5	173.0	A
7710	30	106	—	266.6	—	B
7710	40	108	—	386	—	C
7710	50	109	—	502	—	D
7710	60	113	—	616	—	E
7710	70	111	725	729	735	F

Abb. 11.4.1-3 Prüfprotokoll-Ausschnitt mit Kennsatz-Vorspann und den Prüfschritten 4.1, 4.2, 4.3; gewonnene Meßwerte sind unterstrichen

12. PRÜFABLAUFSPRACHE PRAS

Die PRüfAblaufSprache "PRAS" ist eine streng anwendungsorientierte, interpretative, echtzeitfähige Programmiersprache, die unter dem in Kapitel 10 beschriebenen Betriebssystem ablauffähig ist.

Bei der Prüfablaufsprache PRAS handelt es sich nicht um einen Satz von Spracherweiterungen an einer existierenden Programmiersprache /42/43/, sondern um eine Sprache mit einer eigenständigen Syntax. Diese Eigenentwicklung war notwendig, da die existierenden Prüfsprachen, wie z.B. ATLAS /44/45/46/47/ für die vorgesehenen Anwendungen zu umfangreich sind und mit den zur Verfügung stehenden Mitteln nicht implementiert werden konnten.

PRAS besteht zu einem Teil aus allgemein nutzbaren Sprachelementen, z.B. zur Programmablaufsteuerung oder zur Berechnung arithmetischer Ausdrücke, zum anderen Teil aus Elementen, die sich an den Funktionen der zu bedienenden Prozeßperipherie, im vorliegenden Fall des Kraftstoff-Regler-Prüfstandes orientieren. Diese Funktionen werden repräsentiert durch Statements, die in ihrem Wortlaut (Syntax) weitgehend mit den Begriffen identisch sind, die auch bei manuellem Betrieb des Prüfstandes angewandt werden.

Für die Wahl einer interpretativen High-Level-Programmiersprache waren mehrere Gründen von Bedeutung. Bei der Alternative zwischen Compiler und Interpreter wurde das letztgenannte Verfahren bevorzugt, weil die Entwicklung von Anwenderprogrammen (Programm erstellen, debuggen und modifizieren) auf einem Interpreter durch den interaktiven Betrieb wesentlich vereinfacht und damit beschleunigt werden kann /48/. Bei einem compilierenden Übersetzer sind für eine Programm-Modifikation nötig: Laden des Quellcodes in den Editor, Änderung eintragen, Abspeichern des Quellcodes, Compilieren, Binden, Lokalisieren, Laden des Objektcodes und Starten des Programmablaufs. Gegenüber dieser

relativ aufwendigen Prozedur reduzieren sich die Arbeitsgänge beim Interpreter nach dem Editieren des Quellcodes auf Laden und Starten des exekutierbaren Codes, der in diesem Fall identisch mit dem Quellcode ist. Dies beschleunigt die Programmerstellung insbesondere in der Testphase und erleichtert vor allem solchen Bedienern die Einarbeitung, die wenig Erfahrung im Umgang mit EDV-Anlagen haben. Aus dem gleichen Grund scheidet eine Assembler-ähnliche Sprache /49/ zur Erstellung der Anwenderprogramme aus, da für Assemblerprogrammierung weitgehende Kenntnisse von rechnerinternen Funktionen notwendig sind.

Die relativ niedrige Arbeitsgeschwindigkeit eines Interpreters tritt bei der vorliegenden Implementation in den Hintergrund, da einerseits die zu steuernden Prozesse sehr langsam sind im Verhältnis zur Arbeitsgeschwindigkeit des Rechners (mehrere Größenordnungen), andererseits zeitkritische Funktionen auf separate Prozessoren ausgelagert wurden und somit unabhängig von der Ablaufgeschwindigkeit des Interpreters sind.

12.1 Konzeption als Programmiersprachen-Bausteinsystem

Die Programmiersprache "PRAS" zeichnet sich durch ihren modularen Aufbau und die daraus resultierende Flexibilität ihres Funktionsumfangs aus:

- Der Implementator kann den Funktionsumfang der Sprache selbst definieren und an eine Applikation gezielt anpassen durch Einfügen entsprechender Schlüsselwörter (Keywords) für anwendungsorientierte Ausführungsroutinen (Execution-Subroutine). Diese Routinen sind die Schlüsselelemente des PRAS-Interpreters. Sie transformieren die anwendungsorientierten Befehle in Funktionen der realen Prozeßperipherie.

- Entsprechend den Anforderungen einer Applikation kann der Sprachinterpreter auf Einprozessor- oder Multiprozessor-Konfigurationen implementiert werden.

- Durch Auslagerung zeitkritischer Funktionen auf eigene Prozessoren ist in einem Multi-Prozessor-System eine echte Parallelverarbeitung möglich.

Außerdem läßt das modulare Konzept des Interpreters die Generierung zweier unterschiedlicher Systemversionen zu:

- Eine geteilte Version, bei der Programmerstellung und Programmausführung auf zwei räumlich getrennten Rechnerebenen geschehen, z.B. Programmerstellung (Edition und Debugging) auf einem Zentralrechner mit Massenspeicher und Drucker und Programmexekution auf den dezentralen, prozeßnahen Prüfstandssteuereinheiten.

- Eine integrierte Version mit Programmerstellung und Ablauf auf den prozeßnahen Steuereinheiten.

Die geteilte Version ist immer dann vorteilhaft, wenn eine wirtschaftliche Mehrfachnutzung der Systemresourcen (Massenspeicher, Drucker, Editierterminal) erwünscht und technisch möglich ist, insbesondere dann, wenn auf mehreren Unterstationen identische, auf dem Zentralrechner erstellte Programme ablaufen. Die integrierte Version ist vorzuziehen für weniger komplexe Anwendungsfälle.

Dieser modulare Aufbau der Programmiersprache "PRAS" erlaubt zusammen mit einem standardisierten Einkartenmikrorechner-System und dem Multiprozessor-Betriebssystem eine optimale Anpassung an die zu erfüllende Aufgabe.

12.2 Geteilte Version der Prüfsprache "PRAS"

In der geteilten Version der Prüfsprache "PRAS" werden die Aufgaben "Prüfprogramm-Edition" und "Prüfprogramm-Debuggen"

einerseits sowie "Prüfprogramm-Abarbeitung" andererseits auf zwei getrennte Teile des Prüfstand-Steuerungssystems aufgeteilt.

12.2.1. Editor-Teil "PREDIT"

Der Editor "PREDIT" läuft auf dem Zentralrechner des Prüfstand-Steuerungssystems und dient zur Erstellung des Prüfprogramm-Quellcodes auf einem CR-Terminal.

12.2.1.1 PREDIT-Kommandosprache

PREDIT-Kommandos sind Befehle, die der Benutzer dem Editor über das Konsolgerät erteilt.

Ein Kommando besteht aus einem Kommandowort, dem ein oder mehrere Parameter folgen können. Kommandos werden sofort nach Abschluß der Eingabe (durch CR LF) ausgeführt. Konnte das Kommando erfolgreich ausgeführt werden, meldet sich PREDIT mit *READY* zurück. Trat ein Fehler auf, wird eine Fehlermeldung und danach *READY* auf dem Bildschirm ausgegeben.

Folgende Kommandos stehen zur Verfügung:

- Eingabe des Quellcodes
  zeilenweise mit integriertem Syntaxcheck.
  Bei korrekter Eingabe wird eine komprimierte Version der Zeile generiert und nach Zeilennummer sortiert im Textpuffer abgelegt.

- Listen des Programms:

  Syntax: LIST (devicename) (zeilen)

  1. Bedienerkonsole (Standardzuweisung)
  2. Drucker (:LP:)
  3. Magnetband-Kassette (:P1:)

- Löschen des Programms im Textpuffer

  Syntax: DELETE (zeilen)

- Speichern des Programm-Codes auf Massenspeichern:

  Syntax: SAVE (devicename) (filename)
  Beispiel: SAVE :FØ:PROGRM.NAM
  SAVE :P1:

- Laden eines Programm-Codes von Massenspeichern:

  Syntax: OLD (devicename) (filename)
  Beispiel: OLD :FØ:PROGRM.NAM
  OLD :P1:

Die optionale Ein/Ausgabe über Magnetband-Kassette des angeschlossenen HP-Terminals (vgl. Kap. 8.1) bietet die Möglichkeit zur rechnerunabhängigen Programmerstellung. Dazu kann auf einem externen Rechner oder CR-Terminal mit Magnetband-Kassettenlaufwerk ein Programm in Form eines ASCII-Files erstellt und auf Magnetkassette gespeichert werden. Der File wird von der Kassette in den Textpuffer des Editors gelesen.

Die Syntax für Geräte- und Filenamen entsprechen der Festlegung für RMX80 und ISIS-II der Firma INTEL /27/37/.

12.2.2 Executer-Teil PREX

Der Executer "PREX" ist resident auf der peripherienahen Prüfstand-Steuereinheit implementiert. "PREX" setzt die Befehlsfolge des mit dem Editor "PREDIT" erstellten Prüfablaufprogramms in die Funktionen der Prozeßperipherie (Prüfstand-Funktionen) um.

Die Abarbeitung erfolgt zeilenweise in den Schritten:

- Erkennen des Befehls durch Vergleich des Schlüsselwortes mit einer Liste zulässiger Befehle.

- Sprung in die zugehörige Subroutine mit Parameterübernahme.

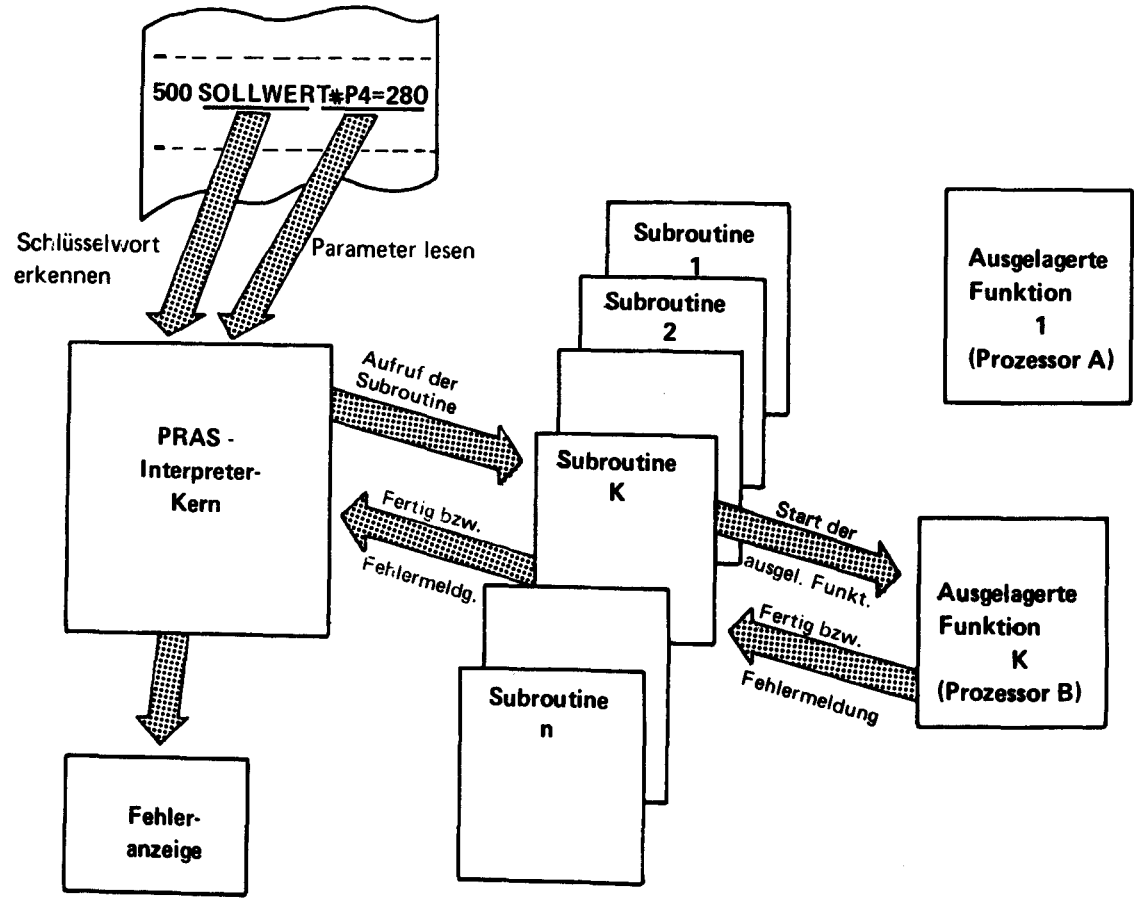


Abb. 12.2.2-1 Abarbeitung eines PRAS-Befehls

- Ausführen der Subroutine.
  Dies kann sowohl die Ausführung einer Subroutine auf dem gleichen Prozessor bedeuten, als auch - bei ausgelagerten anwendungsorientierten Funktionen - die Aktivierung einer Task auf einem externen Prozessor.

- Rückkehr aus der Subroutine in den Interpreterkern ggf. mit Fehleranzeige und Fehlercode.

Danach wird die Programmausführung mit der nächsten Zeile fortgesetzt bzw. eine Fehlerbehandlung durchgeführt.

12.2.3 Debugger-Teil "PREBUG"

Zusätzlich zu den eigentlichen Editierfunktionen bietet "PREDIT" mit dem "DEBUG"-Mode die Möglichkeit, auf dem Zentralrechner die Prüfprogramme ablaufen zu lassen und zu testen.

Die Funktionen des Prüfstandes werden offline simuliert: Die Prozeßperipherie-Schnittstellen werden durch Ein- und Ausgaben über das Bediener-CRT nachgebildet, d.h. der Bediener übernimmt die Funktion des Prüfstandes. Dabei entdeckte Programmfehler können im interaktiven Betrieb korrigiert werden.

Der Debugger "PREBUG" ist als Hilfsmittel zum Test des logischen Programmablaufes zu verstehen. Da kein mathematisches Modell der Prozeßperipherie, bestehend aus Prüfstand und Prüfobjekt, existiert, kann das dynamische Verhalten des Prüfstandes nicht nachgebildet werden.

12.3 Integrierte Version der Prüfsprache "PRAS"

Die integrierte Version der Prüfsprache "PRAS" umfaßt alle Funktionen, die zur Prüfprogrammerstellung, Speicherung und Ausführung auf der Steuereinheit des Prüfstandes benötigt werden.

Die verfügbaren Statements sind unterteilt in die "KOMMANDO"-Statements und "EXECUTION"-Statements. Erstere entsprechen den Editier-Befehlen von "PREDIT" der geteilten "PRAS"-Version, letztere sind identisch mit dem Befehlsvorrat des Executer-Teils "PREX".

Die integrierte "PRAS"-Version bietet neben "KOMMANDO"- und "EXECUTION"-Mode den "IMMEDIATE"-Mode als dritte Betriebsart. Dieser erlaubt eine unmittelbare Ausführung von "EXECUTION"-Statements von der System-Bediener-Konsole aus und ist deshalb beim Debuggen von besonderem Vorteil. Da die integrierte Version eine Kombination der Teile "PREDIT" und "PREX" darstellt, wird auf eine weitere Diskussion an dieser Stelle verzichtet.

12.4 Grundelemente des Prüfsprach-Interpreters "PRAS"

PRAS-Programme bestehen aus einer Folge von Anweisungen. Jede Anweisung belegt eine Programmzeile und besteht aus:

- Zeilen-Nummer (1 - 59999)
- der Anweisung im eigentlichen Sinne (Statement), die mit einem für den Anweisungstyp spezifischen Wort (Keyword) beginnt.

Die Anweisungen werden in aufsteigender Reihenfolge ihrer Zeilennummern im Arbeitsspeicher des Interpreters abgelegt. In der gleichen Reihenfolge verläuft die Abarbeitung, außer bei Sprüngen, bedingten Verzweigungen und Schleifen.

Im folgenden werden nur die für das grundsätzliche Verständnis der Prüfsprache erforderlichen Elemente beschrieben, eine detaillierte Syntax-Beschreibung befindet sich in /50/. Dabei handelt es sich im wesentlichen um die anwendungsorientierten Befehle, die Basiskommunikationselemente für den File-Transfer von und zum zentralen Rechnersystem sowie die Befehle, die über den Bedienfeldprozessor die Verbindung vom Prüfablaufprogramm zum Prüfstandbediener herstellen.

12.4.1 PRAS-Statements

Da die PRüfAblaufSprache "PRAS" als ein Bausteinsystem für anwendungsorientierte Programmiersprachen zu verstehen ist, ist der Funktionsumfang - und damit der Befehlsvorrat - nicht statisch vorgegeben, sondern flexibel den Anforderungen der jeweiligen Applikation anpaßbar. Im folgenden werden die Statements der PRAS-Version 2.0 (1982) beschrieben, die im Rahmen des Projekts implementiert wurde. Die Statements der PRüfAblauf-Sprache "PRAS" sind unterteilt in zwei Gruppen:

a. allgemeine Befehle
b. anwendungsorientierte Befehle

Zu a.
Zu den allgemeinen Befehlen gehören die Statements, die unabhängig von der spezifischen Implementation sind (siehe Tabelle 12.4.1-1).

ANWEISUNG	FUNKTION	BEISPIEL
*	Kommentar	**Beispiel**.
DIM	Zahlenfelddimensionierung	DIM X(20)
RECHNE (auch ohne Schlüsselwort)	Zuweisung	RECHNE A=B+C
WENN	Verzweigung	WENN A=B DANN 500
FUER	Anfang Schleife	FUER J=1 BIS 10
NAECHSTES	Ende Schleife	NAECHSTES J
SPRINGE	Sprung	SPRINGE 500
UNTERPROGRAMM	Unterprogrammaufruf	UNTERPROGRAMM 800
ZURUECK	Rückkehr aus Unterprogramm	ZURUECK
EINGABE	Eingabe über CRT	Eingabe A
DRUCKE	Ausgabe über CRT	DRUCKE "A=";A
SPEICHERE	Variable (in ASCII)	SPEICHERE A(X)=#P0
DATA	Datum (ASCII) einer Variablen zuordnen	DATA XY = B(Z)
LESE	Lesen eines Datenfiles	LESE :F0:DATA.FIL
SCHREIBE	Schreiben eines Datenfiles	SCHREIBE :F0:DATA.FIL
ENDE	Logisches Programmende	ENDE
STOP	Abbruch des Programmablaufs	STOP
KETTE	Laden und Starten	KETTE :F0:NEUES.PRG

Tabelle 12.4.1-1 Allgemeine PRAS Statements

ANWEISUNG	FUNKTION	BEISPIEL
SOLLWERT	Sollwertvorgabe für Regelkreis	SOLLWERT #P0 = 120
RAMPE	Analoge Größe im Rampe verfahren	RAMPE #P0 STOP 140 ZEIT 5
HALTE	Stellglied-Position halten (Regelkreis deaktivieren)	HALTE #P0
MESSE	Istwert einer Analog-Größe erfassen	MESSE #P0
WARTE	Anhalten des Programmablaufes und Warten auf Eingriff des Bedieners	WARTE 10 SEC WARTE AUF FORTSCHALTUNG WARTE AUF EINGABE 3,1 #L0
INFO	Textausgabe an Bediener (mit Display-Steuerung)	INFO 5,0 "DRUCK P0=";#P0
GRAPHIK	Graphische Darstellung auf Video-Display	GRAPHIK 1,0
LPC	Ventilsteuerung	LPC LANE 1
ELPC	Ventilsteuerung	ELPC
SOC1	Ventilsteuerung	SOC1 EIN
SOC2	Ventilsteuerung	SOC2 AUS
STARTERFLOW	Ventilsteuerung	STARTERFLOW ZU
DWV	Ventilsteuerung	DWV EIN
UMSCHALTUNG	Mehrwegeventil-Steuerung	UMSCHALTUNG #P3
CYCLE	Schalterbedienung Cycle-Einrichtung	CYCLE EIN
VAKUUMPUMPE	Schalterbedienung Vakuumpumpe	VAKUUMPUMPE EIN
ALARM	Schnellabschaltung des Prüfstandes	ALARM

Tabelle 12.4.1-2 Implementierte anwendungsorientierte Befehle

Zu b.
Die anwendungsorientierten Statements spiegeln die Peripheriefunktionen der beschriebenen Implementation wider.

Die Schlüsselworte (Keywords) sind die Funktionsbezeichnungen des mit dem Steuerungsrechner betriebenen Prüfstandes (siehe Tabelle 12.4.1-2).

12.4.2 Namen

Namen sind mnemotechnische Bezeichnungen in "PRAS"-Programmen. Sie bezeichnen

- Variablen

  Diese bestehen aus:
  - einem Buchstaben
  - dem ein weiterer Buchstabe oder eine Ziffer folgen kann (nicht muß)

  Ihnen kann vom PRAS-Programm ein Wert zugewiesen werden.

  Beispiel für Namen:

  A
  BC
  D1

- Prozeßvariablen

  Diese bestehen aus:

  - der Prozeßvariablen-Kennung #
  - einem Buchstaben
  - dem ein weiterer Buchstabe oder eine Ziffer folgen muß.

Die Namen der Prozeßvariablen können vom Programmierer nicht beliebig gewählt werden, da ihnen bei der Systemgenerierung eine für die Applikation spezifische Bedeutung zugewiesen wurde und entsprechende, hierfür reservierte Namen in eine Liste eingetragen wurden. Über diese Namen geschieht der Zugriff auf die Prozeßgrößen (Istwerte messen und Sollwerte vorgeben).

Beispiele für Prozeßvariablen-Namen:

```
#P0
#QB
```

Eine Liste der bei der vorliegenden Implementation zugelassenen Prozeßvariablennamen und ihre Bedeutung befindet sich im Anhang.

12.5 Anwendungsorientierte Befehle

12.5.1 SOLLWERT

Mit Hilfe des SOLLWERT-Statements werden die digitalen Regelkreise der Prüfstandsteuerung bedient.

Syntax:

```
SOLLWERT # Prozeßvariable  =  Ausdruck
```

Beispiel:

```
SOLLWERT #P0 = 29.5
SOLLWERT #QB(#IL) = 1.2 * B1
```

Bei Ausführung des SOLLWERT-Statements bestimmt der Interpreter aus der Prozeßvariablen der linken Seite die Regelkreisnummer. Als Prozeßvariable dürfen nur die Regelkreisbezeichnungen (siehe Anhang) verwendet werden.

Das SOLLWERT-Statement dient neben der Vorgabe von Sollwerten auch zur Umschaltung der Regelkreise zwischen Ein-

fach- und Folgeregelung (vgl. Kap. 6). Der Interpreter erkennt den Aufruf eines Folgeregelkreises aus der Regelkreisbezeichnung (z.B. #QB(#IL)). Bei Aufruf eines Folgeregelkreises wird der zugehörige Einfachregelkreis abgeschaltet. Das gleiche gilt für den umgekehrten Fall: Bei Sollwertvorgabe für einen Einfachregelkreis wird, falls vorhanden, der entsprechende Folgeregelkreis deaktiviert.

Der Ausdruck der rechten Seite wird ausgewertet, skaliert, d.h. auf einen Wert zwischen 0 und 4095 normalisiert und als 4-Byte Floating-Point Zahl an das DDC-Modul übergeben. Nach der Übergabe wartet die Ablaufsteuerung auf eine Übernahmequittung des DDC-Modul. Trifft diese nicht innerhalb von 500 msec ein, wird ein hardwaremäßiger Fehler des DDC-Moduls angenommen und eine Meldung auf dem Bediener-CRT ausgegeben.

12.5.2 RAMPE

Das RAMPE-Statement erlaubt das rampenförmige Verfahren des Istwertes eines Regelkreises.

Syntax:

```
RAMPE # Prozeßvariable  STOP  Ausdruck  ZEIT  Zahl
```

Beispiel:

```
RAMPE #P4 STOP 120 ZEIT 10
```

Das RAMPE-Statement kann, wie das SOLLWERT-Statement, benutzt werden, um eine Regelgröße auf einen definierten Istwert zu fahren. Hierzu wird, ausgehend von dem augenblicklichen Istwert, der Sollwert des Regelkreises so verändert, daß nach der durch ZEIT angegebenen Zeitdauer (in Sekunden) der Istwert, der in der Programmzeile dem Kennwort STOP folgt, erreicht ist. Für das RAMPE-Statement benutzt der Interpreter die gleichen Funktionen wie für das SOLLWERT-Statement. Bei Versagen des DDC-Modul wird die gleiche Fehlermeldung ausgegeben.

12.5.3 HALTE

Mit dem HALTE-Statement wird ein Regelkreis deaktiviert, indem der augenblickliche Ausgabewert der Stellgröße festgehalten ("eingefroren") wird.

Syntax:

```
HALTE # Prozeßvariable
```

Beispiel:

```
HALTE #P3
```

Das HALTE-Statement wird benutzt, wenn ein Analogwert von der digitalen Regelung unbeeinflußt bleiben soll.

Die Regelung des Kreises wird wieder aktiviert, wenn der Kreis erneut durch das SOLLWERT- oder RAMPE-Statement angesprochen wird.

Die Fehlererkennung entspricht dem SOLLWERT-Statement.

12.5.4 MESSE

Mit dem MESSE-Statement werden Prozeßgrößen erfaßt und als 4-Byte-Floating-Point-Zahl unter dem der Prozeßgröße entsprechenden Prozeßvariablennamen abgelegt.

Syntax:

```
MESSE # Prozeßvariable
```

Beispiel:

```
MESSE #P0
```

Nachdem die Prozeßvariable durch das MESSE-Statement angelegt wurde, kann auf diese wie auf eine gewöhnliche Variable zugegriffen werden. Die interpreterinternen Auswerteroutinen können nicht unterscheiden, ob eine Variable durch ein RECHNE- oder ein MESSE-Statement angelegt wurde.

12.6 Dezentraler Plattenzugriff (vgl. 10.3.4)

12.6.1 SCHREIBE

Mit dem SCHREIBE-Statement werden die durch das PRAS-Programm gewonnenen Daten in einen Magnetplatten-File des Zentralrechners geschrieben.

Syntax:

SCHREIBE Prüfschritt-Nummer

Beispiel:

SCHREIBE 4.1

Zuvor unter der gleichen Prüfschritt-Nummer im selben File abgelegte Daten werden überschrieben. Positionen des Datenfiles, die nicht mit dem SPEICHERE-Statement beschrieben wurden, sind mit Blanks gefüllt.

12.6.2 LESE

Das LESE-Statement lädt Daten aus einem Plattenfile des Zentralsystems über die Kommunikationsstrecke in einen reservierten Puffer.

Syntax:

LESE Prüfschritt-Nummer

Beispiel:

LESE 4.3

Es werden jeweils die Daten eines Prüfschrittes geladen. Die Daten werden als ASCII-Strings in den Puffer geschrieben. Sie können von dort mit Hilfe des DATA-Statements in das Floating-Point-Format umgewandelt und als Variablen im Variablenpuffer abgelegt werden.

12.7 WARTE

Das WARTE-Statement dient zum Anhalten des Programms. Es kann mit drei verschiedenartigen Parametern aufgerufen werden

Syntax:

WARTE Zeitangabe in Sekunden

WARTE Wiederanlaufbedingung

Beispiel:

a. WARTE 10 SEC

b. WARTE AUF FORTSCHALTUNG

c. WARTE AUF EINGABE ‹3,1› #L1

Zu a.

Die Programmausführung wird für eine definierte Zeit angehalten und nach Ablauf dieser Zeit mit der nächsten Programmzeile fortgesetzt. Die Zeitangabe muß eine Konstante sein.

Zu b.

Die Programmausführung wird angehalten, bis der Bediener durch Drücken der Taste "1" der Bedienertastatur die Fortsetzung des Programms anfordert.

Zu c.

Die Programmausführung wird angehalten. Über die Steuerzeichen in der eckigen Klammer wird auf dem Bedienerbildschirm eine Eingabe-Bildmaske eingeblendet und vom Bediener die Eingabe eines Zahlenwertes über die Tastatur erwartet. Nach der Eingabe wird der Programmablauf fortgesetzt.

12.8 INFO

Das INFO-Statement dient zur Ausgabe von Bildschirm-Steuerzeichen und Klartextzeilen (Bedienerinformationen) auf dem Bildschirm der Prüfstandsteuerung

Syntax:

INFO Steuerzeichen1, Steuerzeichen2 (Text)

Beispiel:

a. INFO ⟨1,0⟩

b. INFO ⟨5,0⟩ "BEDIENER-MITTEILUNG"

c. INFO ⟨5,0⟩ "DRUCK #P4 =";2.3*Z1;" PSI"

Zu a.

Ein INFO-Statement ohne Text hinter den Steuerzeichen dient ausschließlich zur Bildschirmsteuerung. Eine Tabelle der zulässigen Steuerzeichen und ihrer Funktionen befindet sich im Anhang.

Zu b.

An der über die Steuerzeichen ausgewählten Stelle des Bildschirms wird eine Textzeile (schwarze Schrift auf blauem Hintergrund) ausgegeben.

Zu c.

Textausgabe wie unter b., jedoch wird an der Stelle zwischen den ";" der Zahlenwert des arithmetischen Ausdrucks eingefügt. Der Wert wird mit einer Stelle hinter dem Dezimalpunkt dargestellt.

12.9 GRAPHIK

Das GRAPHIK-Statement dient zur Ausgabe von Bildschirm-Steuerzeichen.

Syntax:

GRAPHIK Steuerzeichen1 , Steuerzeichen2

Beispiel:

GRAPHIK ⟨1,0⟩

Das GRAPHIK-Statement ersetzt das INFO-Statement an den Stellen, wo INFO ausschließlich zur Bildschirmsteuerung

benutzt wird. Im Gegensatz zu INFO sind bei GRAPHIK außer den Steuerzeichen keine weiteren Parameter zugelassen, insbesondere keine Textstrings.

12.10 Statements zur Bedienung binärer Funktionen der Prüfstandsteuerung

Unter binären Funktionen sind bei der vorliegenden Implementation Relais und Ventile zu verstehen, d.h. Komponenten mit zwei definierten Schaltzuständen.

Für die Statements wurden die gleichen Begriffe gewählt, die in der Fachsprache angewendet werden. Die Begriffe sind zum Teil in englischer Sprache, da die damit bezeichneten Teile britische Konstruktionen sind, die auch in der Umgangssprache der deutschen Fachleute mit den englischen Bezeichnungen beschrieben werden.

Bei Bedienung der binären Komponenten überwacht die Prüfstandsteuerung die Umschaltung durch Abfrage eines Rückmeldesignals. Trifft die Rückmeldung nicht innerhalb einer definierten Zeit ein, wird eine Funktionsstörung angenommen, eine Fehlermeldung auf dem Bedienerbildschirm ausgegeben und der Prüfstand abgeschaltet.

12.10.1 LPC

Mit dem LPC-Statement werden zwei Ventile, LPC LANE 1 und LPC LANE 2, bedient.

Syntax:

```
LPC  Schaltzustand
```

Beispiel:

```
LPC LANE 1
LPC LANE 2
```

Die Schaltzustände LANE 1 und LANE 2 sind invers, d.h. das Einschalten der einen Funktion impliziert das Ausschalten der anderen.

12.10.2 ELPC

Der ELPC-Befehl bewirkt das Einschalten des Ventils ELPC (Emergency LPC).

Syntax:

```
ELPC
```

Beispiel:

```
ELPC
```

Dieses Statement hat keine weiteren Parameter.

Durch Betätigung eines LPC-Ventils (Lane 1 oder Lane 2) wird das Ventil ELPC abgeschaltet.

12.10.3 SOC

Mit dem SOC-Statement werden zwei Ventile, SOC 1 und SOC 2, bedient.

Syntax:

```
SOC  Ventil-Nummer    Schaltzustand
```

Beispiel:

```
SOC 1 EIN
SOC 1 AUS

SOC 2 EIN
SOC 2 AUS
```

Die beiden Funktionen SOC 1 und SOC 2 sind invers, d.h. das Einschalten der einen Funktion impliziert das Ausschalten

der anderen. Beide Funktionen können jedoch gleichzeitig ausgeschaltet sein.

12.10.4 STARTERFLOW

Das STARTERFLOW-Statement schaltet das Ventil STARTERFLOW.

Syntax:

```
STARTERFLOW   Schaltzustand
```

Beispiel:

```
STARTERFLOW AUF
STARTERFLOW ZU
```

12.10.5 DWV

Mit dem DWV-Statement wird das Ventil DWV angesteuert.

Syntax:

```
DWV   Schaltzustand
```

Beispiel:

```
DWV  EIN
DWV  AUS
```

12.10.6 UMSCHALTUNG

Mit dem UMSCHALTUNG-Statement wird das Ventil zur Umschaltung des Meßwertgebers im Regelkreis #P3 / #P3A bedient.

Syntax:

```
UMSCHALTUNG   Schaltzustand
```

Beispiel:

```
UMSCHALTUNG  #P3
UMSCHALTUNG  #P3A
```

12.10.7 CYCLE

Das CYCLE-Statement bedient die CYCLE-Einrichtung.

Syntax:

CYCLE Schaltzustand

Beispiel:

```
CYCLE EIN
CYCLE AUS
```

12.10.8 VAKUUMPUMPE

Mit dem VAKUUMPUMPE-Statement wird die Vakuumpumpe geschaltet.

Syntax:

VACUUMPUMPE Schaltzustand

Beispiel:

```
VAKUUMPUMPE EIN
VAKUUMPUMPE AUS
```

12.11 ALARM

Das ALARM-Statement bewirkt die Schnellabschaltung des Prüfstandes.

Syntax:

ALARM

Beispiel:

```
ALARM
```

Die binären Funktionen werden in einen sicheren Zustand geschaltet (z.B. die Spulen elektromagnetischer Ventilantriebe stromlos) und alle Regelkreise auf den Sollwert 0.0 gefahren.

13. SCHLUSSBETRACHTUNG

Das beschriebene Automatisierungssystem befindet sich seit etwa 6 Monaten im Probebetrieb. Dabei hat sich die Flexibilität der Prüfsprache "PRAS" als eine besonders herausragende Eigenschaft erwiesen, da auch die trotz sorgfältiger Systemanalyse zunächst verdeckten, aber vom Anwender später gewünschten Eigenschaften des Automatisierungssystems in sehr kurzer Zeit implementiert werden konnten. Der vollautomatische Prüfablauf und die damit mögliche Entlastung der Prüfstandsoperateure von extremen Umweltbelastungen führte schnell zu einer weitgehenden Akzeptanz. Die Verkürzung von Meßzeiten wird in Zusammenhang mit der durch die automatischen Prüfabläufe bedingten Mehrfachbedienungsmöglichkeit von Prüfständen zu den erhofften Rationalisierungseffekten führen. Die leichte Erstellung von Prüfprogrammen kann auch der Einführung einer überlagerten Prüfebene dienen, die aus Kostengründen in der Prüfvorschrift (Test schedule) zunächst nicht vorgesehen war.

Das auf Basis eines kommerziell verfügbaren Kerns entwickelte Betriebssystem unterstützt im Gegensatz zu inzwischen auf dem Markt angebotenen Systemen die transparente Kommunikation zwischen räumlich verteilten Mikrorechnerclustern. Der Aufbau eines Multiprozessorechtzeitbetriebssystems erfordert selbst bei der hier vorgenommenen Integration eines bereits entwickelten Kerns einen hohen Kenntnisstand, zumal der verfügbare Betriebssystemkern keine Schutzmechanismen gegen unzulässige Interaktionen zwischen den Anwendertasks bzw. zwischen Anwerndertasks und Executive hat. Darüber hinaus haben sich das Fehlen von adäquaten Testhilfen und eine effektive Unterstützung bei der Systemgenerierung zunächst als besonders nachteilig erwiesen. Moderne leistungsfähige Prüfmittel wie "In Circuit Emulatoren", die bei der Realisierung von Hard- und Software für Monoprozessorsysteme ein hervorragendes Werkzeug sein können, sind in einer Mehrprozessorumgebung nur bedingt von Nutzen. Insbesondere beim Test der Interprozes-

sor-Kommunikation hat sich der Versuch, zwei Emulatoren auf zwei Mikrorechnerkarten einzusetzen, als wenig erfolgreich erwiesen, da eine Synchronisation zwischen den Emulatoren nicht möglich ist. Ein Test-Softwarepaket (Active Debugger), das Bestandteil des RMX80 Pakets ist, konnte ebenfalls beim Testen der Interprozessorkommunikation nur bedingt eingesetzt werden, da auch hier die benötigten Funktionen nicht zur Verfügung stehen und die Zurückverfolgung der Programmausführung (Tracing) nicht gegeben ist. Ein besonderes Problem ergibt sich aus der Notwendigkeit, in einer Multiprozessorumgebung die Programmsysteme der einzelnen Mikrorechner auch gegeneinander zu schützen, um eine fundierte, gesicherte Fehlersuche zu ermöglichen. Dabei wären auch hardwaremäßige Voraussetzungen wünschenswert, die die Gültigkeit von Botschaften zwischen den Prozessoren ohne Zeitverlust prüfen können /51/. Es ist zu erwarten, daß bei weiterer Verbreitung von Multi-Mikrorechnersystemen auch effizientere Werkzeuge zum Programmtest von den Herstellern angeboten werden. Dabei wird die flankierende Einführung von "silikonisierten" Schutzmechanismen schon auf den eingesetzten Mikroprozessoren selbst /52/ den nicht unerheblichen Anteil der Test- und Prüfkosten an den gesamten Softwareaufwendungen senken helfen.

Wenn auch der Einsatz von 8 Bit µ-Prozessoren selbst wegen ihrer geringeren Leistungsfähigkeit für den beschriebenen Anwendungsfall nicht optimal ist, so bringt doch die Integration von 8 Bit Prozessoren im Rahmen des Mehrprozessorsystems eine beträchtliche Leistungssteigerung. Diese konnte durch die Verwendung von Arithmetik-Koprozessoren und die Erweiterung der lokalen Speicher noch weiter erhöht werden. Die implementierten Speichererweiterungen haben in den meisten Fällen erst die Integration von Funktionseinheiten bezogen auf eine Mikrorechnerkarte und damit die gewünschte Teilbarkeit der Aufgaben in unabhängige Arbeitspakete für mehrere Entwickler ermöglicht.

Verglichen mit einer ersten Analyse der verfügbaren Multiprozessorsysteme und den zugehörigen Bussystemen /15/ hat sich während der Projektlaufzeit keine signifikante Vereinheitlichung in der Angebotspalette ergeben. In jüngster Zeit haben sich in internationalen Arbeitskreisen (z.B. EWICS, European Workshop on Industrial Computer Systems) oder auch auf Grundlage von Absprachen zwischen mehreren Halbleiterherstellern /53/ weitere Bussysteme für Mehrrechneranwendungen qualifiziert, die auch den Einsatz von modernen 32 Bit Mikroprozessoren vorsehen und zu einem Teil sogar Betriebssystemfunktionen unterstützen /54/55/. Ein breites Angebotsspektrum der Herstellerindustrie ist jedoch erst in den nächsten Jahren zu erwarten. Allerdings dürften nach aller Erfahrung einfach zu handhabende zugeschnittene "Software-Tools" auch in absehbarer Zeit nicht zur Verfügung stehen, die die angebotenen Hardwaremöglichkeiten in voller Tiefe ausschöpfen.

In Abb. 13-1 ist das bei der Firma PIERBURG im Einsatz befindliche Produktionssteuerungssystem dargestellt. Zur Zeit sind 10 numerisch gesteuerte Werkzeugmaschinen im Rahmen eines DNC-Systems über eine CAMAC serielle Ringleitung /56/ mit einem Fertigungsrechner verbunden. Insgesamt 15 Betriebsdatenerfassungsstationen mit Ausweisleser unterstützen die schritthaltende Auftragszeiterfassung und on-line Disposition /14/. Ein übergreifendes Planungs- und Steuerungssystem hat über die dargestellte Kopplung zwischen Betriebsrechner und Fertigungsrechner eine direkte Anbindung an den Produktionsbereich.

Um den Aufbau eines globalen Qualitätssicherungssystems zu unterstützen, werden in Zukunft auch die entwickelten Prüfstandsteuereinheiten über das Zentralrechnersystem an eine Unterstation der CAMAC-seriellen Ringleitung angekoppelt. Dabei ist der Aufbau einer Meßdatenarchivdatei vorgesehen, die als Basis für die Nachprüfung von Testtoleranzen, Trendanalysen nach verschiedenen Schlüsseln und Generierung von Qualitätsreports dienen soll. Darüber hinaus ist eine Rückmeldung von Fehlern und Ausprägungen

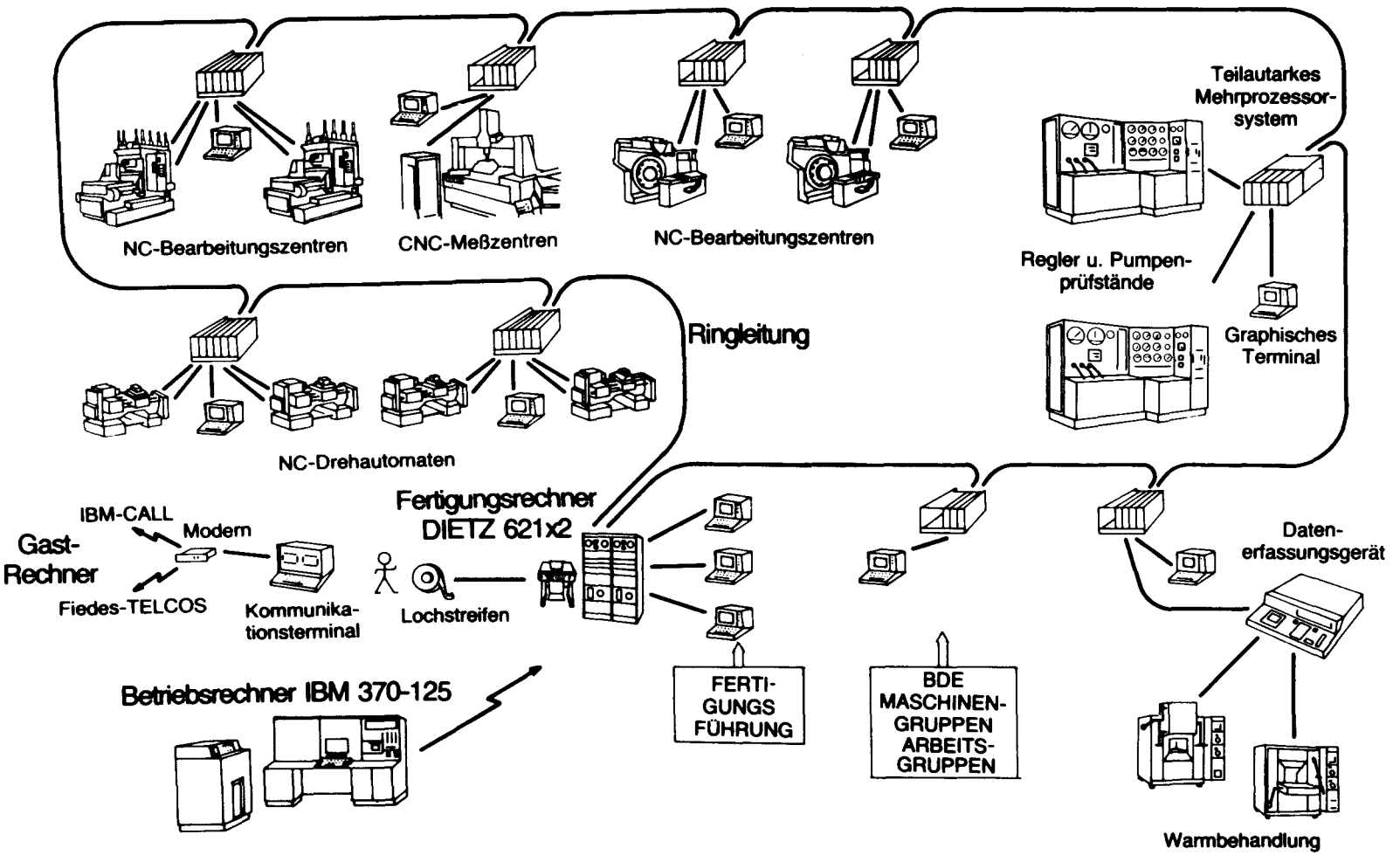


Abb. 13-1 Rechnerunterstütztes Produktionssteuerungssystem

von Qualitätsmerkmalen an die Planungs- und Steuerungsebene vorgesehen, um den Qualitätsregelkreis schließen zu können (vgl. /57/).

Bei der Implementation des Systems zur direkten numerischen Steuerung (DNC) und der Betriebsdatenerfassung hat die Firma PIERBURG im Rahmen von Zusammenarbeitsverträgen zu einem Teil auf Entwicklungsergebnisse zurückgegriffen, die im wesentlichen von Mitarbeitern des Zentrallabors für Elektronik der KFA Jülich und dem Werkzeugmaschinenlabor der TH Aachen im Rahmen eines PDV-Projektes /58/ erarbeitet wurden.

Auch die Automatisierung der Kraftstoffreglerprüfstände wurde von der KFA Jülich maßgebend unterstützt. Bei allen nunmehr abgeschlossenen Projekten hat sich gezeigt, daß insbesondere für Unternehmen mittlerer Größenordnung die Zusammenführung von Know How aus Forschungszentren und Hochschule mit dem Anwender in der Industrie zu einer zukunftsorientierten Problemlösung wesentlich beigetragen hat, die auch den Weg für eine weitere Vermarktung durch Beachtung von Standardisierungsaspekten und Anwendung innovativer Technologien öffnet.

14. LITERATUR

/ 1 / Möhl, R.:
Rechnergestützte Qualitätsdatenerfassung,
Dissertation, TH Aachen, Febr. 1980

/ 2 / Autorenkollektiv:
Automatisches Messen und Prüfen in der Fertigung,
Referate 12. IPA-Arbeitstagung, Stuttgart,
24. - 26. Juni 1980

/ 3 / Zwoll, K., Hendrix, V., Pothen, B., Schmidt, J.:
Rechnergeführte Automatisierung von Regelgeräte-Prüfsystemen,
Spezielle Berichte der KFA-Jülich, Jül-Spez-51,
Juli 1979

/ 4 / VDE-Vorschrift:
VDE 0165 / 6.80,
Errichten elektrischer Anlagen im explosionsgefährdeten Bereich, Juni 1980

/ 5 / Intel:
iSBC Microcomputer Systems Configuration Guide,
Datenblatt B72-679
Intel Corp., 1979

/ 6 / Intel:
MULTIBUS-Specifications
Publ. Nr. 98 00 683-04
Intel Corp., 1982

/ 7 / Bieck, R. D.:
AMS, das erste Baugruppensystem aus Europa für Multicomputer-Anwendungen,
Siemens Bauteile Report 17 (1979) Heft 2, S. 29 - 32

/ 8 / Siemens:
Microcomputer-Baugruppensystem AMS
Systemübersicht
Best.Nr. B/2634
Siemens AG, Feb. 1982

/ 9 / Siemens:
AMS Computer-Baugruppensystem, Systembeschreibung,
Best.Nr. B/2773
Siemens AG, Juli 1982

/ 10 / Siemens:
Realtime-Betriebssystem Kern SMP - RTOS - A11,
Technische Beschreibung,
Siemens AG, Okt. 1980

/ 11 / Hewlett-Packard:
Intelligent Graphics Terminal 2647A
User's Manual,
Manual Nr. 02647-90001
Hewlett-Packard Co., Febr. 1979

/ 12 / Hewlett-Packard:
Graphics Printer 2631
User's Manual
Manual Nr. 02631-90001
Hewlett-Packard Co., Apr. 1979

/ 13 / Abbott, D., Schmidt, J., Zwoll, K.:
Serial Communication in a Multi-Microcomputer Environment,
Interner Bericht des Zentrallabors für Elektronik der Kernforschungsanlage Jülich, KFA-ZEL 501-582,
Dez. 1982

/ 14 / Billen, F., Zwoll, K.:
Ein hochstandardisiertes Prozeßsteuerungssystem zur rechnergestützten Produktionssteuerung einer Kraftstoffreglerfertigung,
Realtime-Data-Handling and Process Control,
H. Meyer, North-Holland Publ. Com., Brüssel and Luxemburg, 1980

/ 15 / Zwoll, K., Müller, K. D., Schmidt, J.:
A Multi-Processor-Approach for Automation of Quality Control in an Overall Production Control Systems,
Proceedings of the IFAC Workshop: Distributed Computer Control Systems, Tampa, Florida, USA, Oct. 1979,
Pergamon Press, Oxford Ox3 OBW, England

/ 16 / HMW Data System:
Color Graphik Display MCD 4001
Handbuch
HMW DATA SYSTEM GmbH, München, 1980

/ 17 / Siemens:
PLM-Programmiersprache,
Handbuch, Best.Nr. B 1838
Siemens AG, April 1979

/ 18 / Intel:
8080 / 8085 Assembly Language Programming Manual,
Manual Nr. 98 00 301 C
Intel Corp. 1977

/ 19 / Dijkstra, E. W.:
Solution of a Problem in Concurrent Programming Control,
Communications of the ACM,
Vol. 8, Nr. 9, Sept. 1965, S. 569

/ 20 / Dijkstra, E. W.:
Cooperating Sequential Process, in:
Programming Languages, Accademic Press, New York,
1968, S. 43 - 112

/ 21 / Lagally, K.:
Synchronisation in a Layered System, in:
Operating Systems, an advanced Course,
Springer Verlag, Berlin, 1978, S. 253 - 281

/ 22 / Hansen, B.:
Distributed Process: A Programming Concept,
Comm. ACM, Vol. 21, Nr. 11, Nov. 1978, S. 934 - 941

/ 23 / Bloom, T.:
Evaluating Synchronization Mechanisms,
Proc. 7th Symposium on Operating System Principles,
ACM, Dez. 1979, S. 24 - 32

/ 24 / Burgett, K., O'Neil, E. F.:
An Integral Real-Time Executive for Microcomputers,
Computer Design, Juli 1977, S. 77 - 82

/ 25 / Autorenkollektiv:
On Operating Systems,
Industrial Programming Inc., New York, USA, 1979

/ 26 / Chien, Y. P.:
Multi-Tasking Executive Simplifies
Real-Time System Design,
Computer Design, Jan. 1980, S. 109 - 118

/ 27 / Intel:
Introduction to the RMX80/88 Real Time Multitasking
Exectives,
Manual Nr. 143238-002,
Intel Corp., 1977

/ 28 / Jurca, I.:
A Multiprocessor System with Multitasking Facilities,
Proefschrift, Technische Hogeschool Delft, Juni 1977

/ 29 / Meißner, K.:
Ein homogenes, modulares Multi-Mikroprozessorsystem für Realtimeanwendungen,
Dissertation, Ruhr-Universität Bochum, Okt. 1978

/ 30 / IPI, Industrial Programming Inc.:
MTOS, Real Time - Multi-Tasking/Multi-Processor Operating System,
SCS, Scientific Control System GmbH, 2 Hamburg 62, Oehleckerring 40, SCS-Form 0399-1 (0882)

/ 31 / Siemens:
AMS-RMOS 1M-A61/D
Echtzeit-Multitasking und Multicomputer-Betriebssystem (Handbuch)
Best. Nr. C8256-A-D7
Siemens AG, 1980

/ 32 / Kuhn, K.:
Mehrrechner- und echte Multiprozessor-Systeme mit fertigen µP-Platinen,
Elektronik 1979, Heft 8, S. 77 - 80

/ 33 / Pruitt, J. L., Case, W. W.:
Architecture of a Real-Time Operating System,
Proc. 5th Symposium on Operating System Principles,
ACM, Nov. 1975, S. 51 - 59

/ 34 / Lalive d'Epinay, Th.:
Structure of an Ideal Distributed Computer Control System,
Distributed Computer Control Systems,
Proc. of the IFAC Workshop, Okt. 1979, S. 65 - 73

/ 35 / Intel:
RMX80 User's Guide,
Manual Nr. 9800522B,
Intel Corp., 1977

/ 36 / Ritchie, D. M., Thompson, K.:
The UNIX Time-Sharing System,
Comm. ACM, Vol. 17, Nr. 7, Juli 1974, S. 365 - 375

/ 37 / Intel:
ISIS-II User's Guide
Manual Nr. 9800306-06
Intel Corp., 1980

/ 38 / Digital Research:
CP/M 2.Ø Interface Guide,
Digital Research Corp., 1979

/ 39 / Latzel, W.:
Regelung mit dem Prozeßrechner (DDC),
Wissenschaftsverlag Bibliographisches Institut,
Zürich, 1977

/ 40 / Takahashi, Y. et al:
Parametereinstellung bei linearen DDC-Algorithmen,
Regelungstechnik und Prozeßdatenverarbeitung, Heft 6,
1971, S. 237 - 244

/ 41 / Schmidt, J., Pofahl, E.:
Digitale Regelung mit Multi-Mikrocomputersystem,
Interner Bericht des Zentrallabors für Elektronik
der Kernforschungsanalge Jülich, KFA-ZEL-500-379,
Mai 1979

/ 42 / Dietz Computer Systeme:
Dietz 621 CAMBAS-Beschreibung
Handbuch Nr. 2-7806-01-108,
Dietz Computer Systeme, 4330 Mülheim, Solinger Str. 9

/ 43 / Brandenburg, G., Halling, H., Lech, G., Ringel, H.:
Realtime BASIC for MACAMAC (INTEL 8080)
Interner Bericht des Zentrallabors für Elektronik
der Kernforschungsanlage Jülich, KFA-ZEL-500-177,
Jan. 1977

/ 44 / Kurkjian, R. G.:
IEEE Standard 416 (ALTAS),
Progress Report,
Conference Record AUTOTESTCON '78
28. - 30 Nov. 1978, San Diego, CA, USA

/ 45 / Loveman, D. B.:
Language Design for Automatic Test Equipment,
Conference Record AUTOTESTCON 1977, 2. - 4. Nov. 1977,
Hyannis, MA, USA

/ 46 / Grady, R.:
A look at a commercial ATLAS Compiler,
1977 Electro Conference Record, 19. - 20. April 1977,
New York, USA

/ 47 / Easton, I. G.:
IEEE Adaption of the ATLAS Test Language,
Electro Conference Record, 1977,
19. - 20. April 1977, New York, USA

/ 48 / Crowley-Milling, M. C., Shering, G. C.:
The NODAL-System for the SPS
CERN-Report 78-07, Sept. 1978,
Geneva, Switzerland

/ 49 / Kroneberg, A.:
Vergleich von Übersetzungsverfahren,
Elektronik, Heft 10, 1981, S. 84 - 86

/ 50 / Schmidt, J., Zwoll, K.:
Anwendungsorientierte Echtzeit-Programmiersprache
PRAS,
Interner Bericht des Zentrallabors für Elektronik
der Kernforschungsanlage Jülich, KFA-ZEL-501-383,
Dez. 1982

/ 51 / ESONE Small System Standard Group:
Proposal for a Standard System Specification E3S,
Tagungsbeitrag ESONE AGA, Berlin, 1983

/ 52 / Intel:
Introduction to the iAPX 286
Manual Nr. 210308-001
Intel Corp., 1982

/ 53 / Motorola:
VME bus Specification Manual
Manual Nr. M68KVMEB (D1)
Motorola Inc., Oct. 1981

/ 54 / Borrill, P.:
IEEE P 896 - the Futurebusproject,
Microprocessors and microsystems,
Vol. 6 No. 9, Nov. 1982, S. 489 - 495

/ 55 / Kirrmann, H. et al.:
I 896 - A proposed Standard Backplane Bus
Specification for Advanced Microcomputer Systems,
Proposal of the I 896 Working Group of
EWICS - TC 10 2 ,
Draft 5.1, Nov. 1982

/ 56 / Erven, W., Zwoll, K.:
Ein Prozeßdatenübertragungssystem mit standardisierten Komponenten,
Berichte der Kernforschungsanalage Jülich,
Jül-1668, August 1980

/ 57 / Zwoll, K.:
Vorprojektstudie zur rechnerunterstützten Qualitätssicherung mit Schwerpunkt "Prüfdatenerfassung in einem Informationsverbund"
Febr. 1981, Interner Report, Pierburg Luftfahrtgeräte Union GmbH, Neuss

/ 58 / Zenner, K., Zwoll, K.:
Modulares DNC-System mit standardisierten Komponenten,
Technische Rundschau, Bern, Nr. 21, 23. Mai 1978

15. ANHANG

A. Beispiel Prüfschritte 5.0, 6.2, 6.3

Die für die Abnahme eines Flugkraftstoffreglers durchzuführenden Prüfungen sind in der Prüfvorschrift (test schedule) festgelegt. Die Gesamtprüfung besteht aus 49 Prüfschritten, von denen 42 als Funktionsprüfungen auf dem Prüfstand durchgeführt werden. Im folgenden werden drei Prüfschritte beschrieben, die als exemplarisch für die bei der Abnahmeprüfung durchzuführenden Prüfungen gelten können.

Prüfschritt 5.0:
Vollautomatischer Ablauf ohne Bedienereingriff.

Prüfschritt 6.2:
Teilautomatischer Ablauf mit definierten Haltepunkten für Bedienereingriffe.

Prüfschritt 6.3:
Vollautomatischer Ablauf mit abnahmekritischen Prüfpunkten.

Zu den genannten Prüfschritten befinden sich Struktogramme, Programmlistings und Prüfprotokolle im Anschluß an die folgenden Kurzbeschreibungen.

Zu Prüfschritt 5.0:

In Prüfschritt 5.0 wird die Abhängigkeit zweier Prozeßgrößen, des Differenzdruckes PDS-VMO und des Kraftstoffflusses Q-Burner von einer dritten Prozeßgröße, dem Luftdruck KP4P festgestellt.

Nachdem die Prüfstand-Grundwerte, das sind die Werte der Prozeßgrößen zu Beginn einer Prüfung, ihre Sollwerte erreicht haben und sich ein stationärer Zustand eingestellt hat, wird der Druck KP4P über den Einstellbereich von 20 bis 70 psi in Stufen von 10 psi verfahren. Nach dem Er-

reichen jeder Stufe werden die Prozeßgrößen PDS-VMO und Q-Burner gemessen und abgespeichert.

Die in Prüfschritt 5.0 gemessenen Werte für den Fluß Q-Burner werden mit denen aus einem früheren Teil des Prüfablaufs, dem Prüfschritt 4.3, verglichen. Übersteigt die Differenz zwischen den Meßwerten aus Prüfschritt 4.3 und Prüfschritt 5.0 eine maximal zulässige Grenze, muß dies durch eine mechanische Nachbearbeitung korrigiert und die Prüfung (Prüfschritt 4.3 und 5.0) wiederholt werden.

Zu Prüfschritt 6.2:

In Prüfschritt 6.2 wird eine Kennlinie des Kraftstoffreglers kalibriert, die den Zusammenhang zwischen einem Steuerstrom (I-LPC) und dem Kraftstoffffluß bestimmt. Der Verlauf der Kennlinie wird über drei Kalibrierschrauben im unteren, mittleren und oberen Teil des Stellbereichs festgelegt. Die Kalibrierschrauben müssen manuell vom Bediener verstellt werden. Die Endeinstellungen der drei Kalibrierschrauben werden interativ erreicht. Das Programm führt den Bediener im Dialogbetrieb durch die Folge von Einstelltätigkeiten. Sind die Einstellungen abgeschlossen, wird der gesamte Stellbereich des Stromes noch einmal durchfahren. Die Meßwerte werden gespeichert und auf zulässige Toleranzen überprüft. Sollten einzelne Toleranzen nicht erreicht werden, muß die Kalibrierung noch einmal durchlaufen werden.

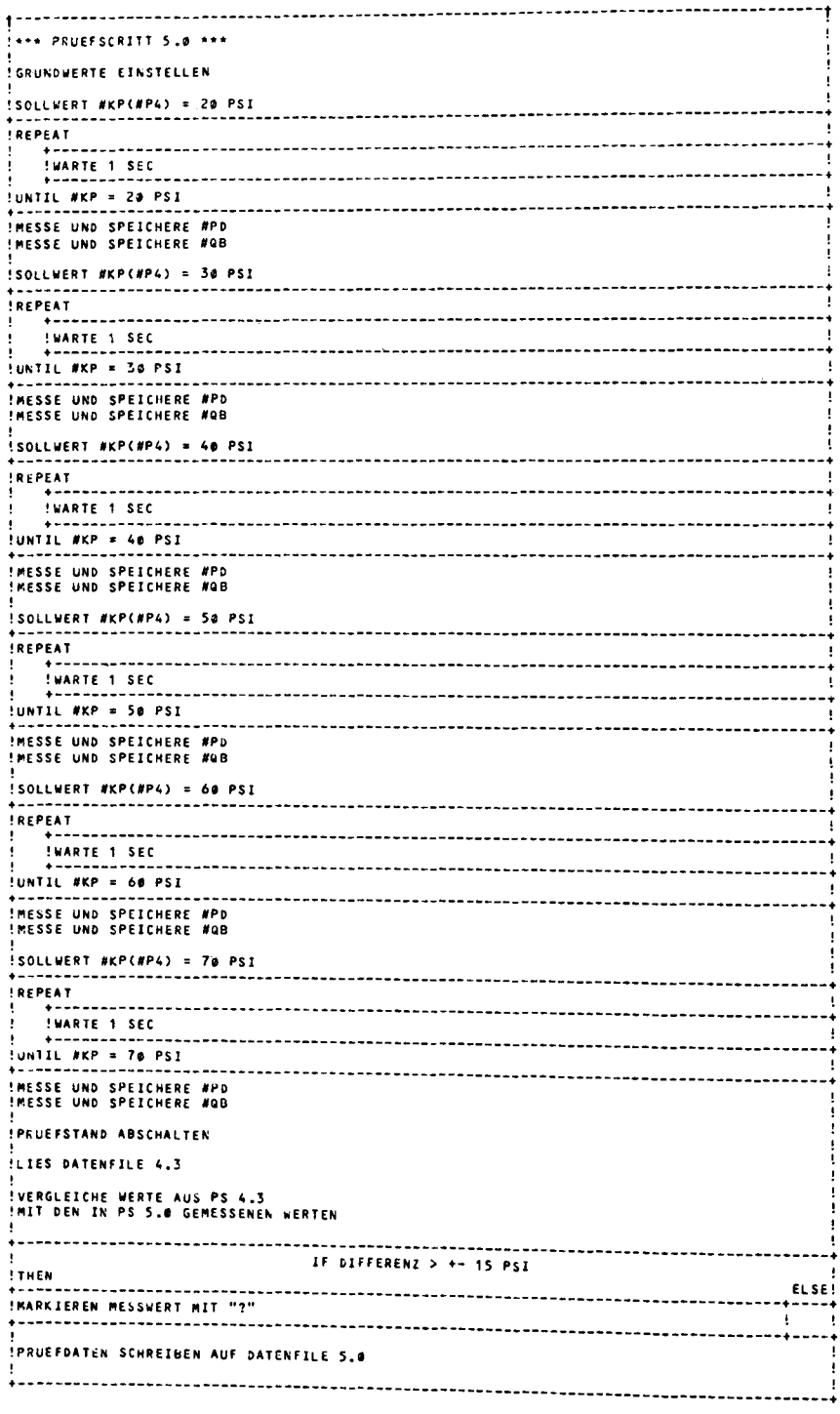
Zu Prüfschritt 6.3:

Nach der Kalibrierung der I-LPC-Kennlinie in Prüfschritt 6.2 wird in Prüfschritt 6.3 das Verhalten des Prüflings im Betrieb mit steigendem und fallendem Kraftstoffffluß getestet.

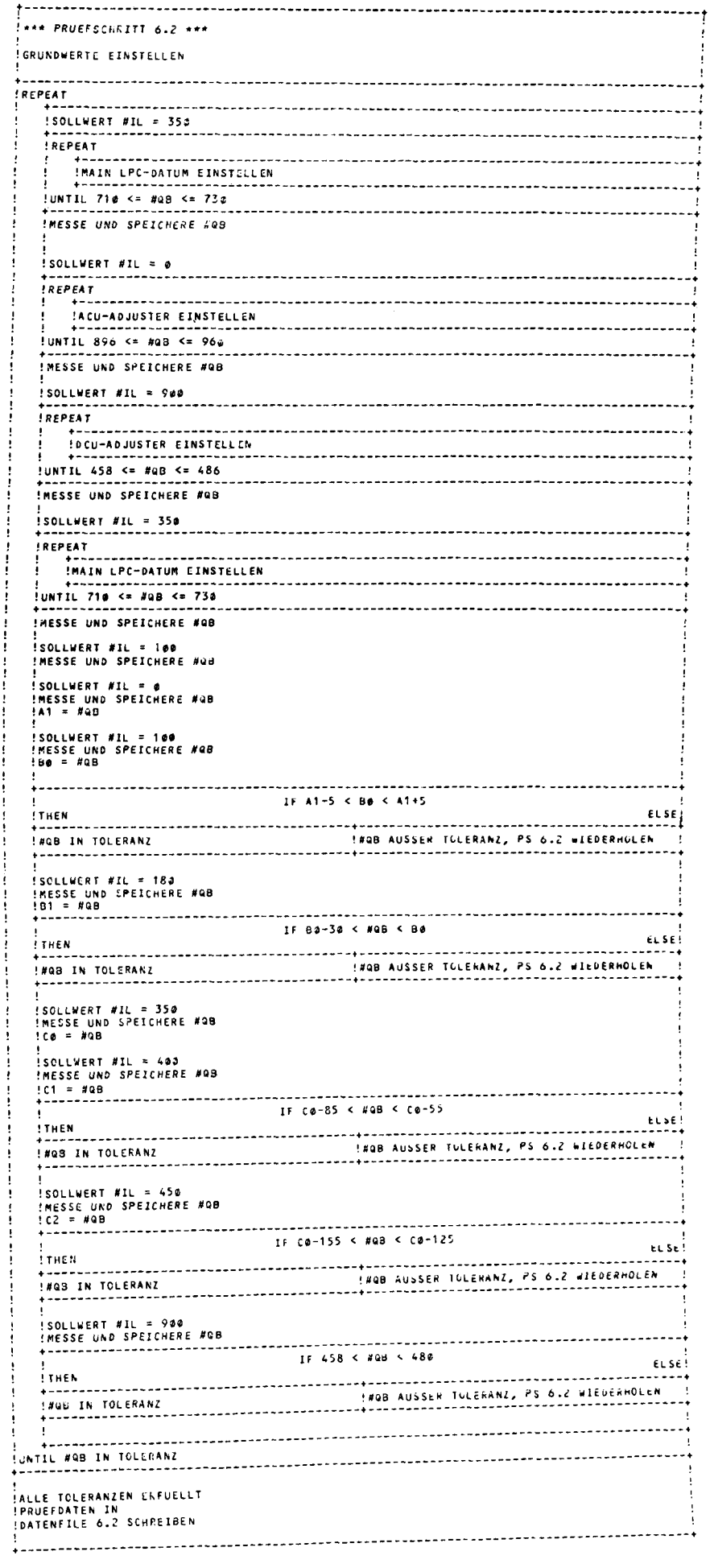
Im ersten Teil der Prüfung wird der Steuerstrom I-LPC von 0 mA erhöht, bis der Kraftstoffffluß einen definierten Wert (hier 800 GpH) erreicht hat. Der Wert des hierbei

eingestellten Stroms wird gespeichert. Danach wird der Strom so variiert, daß sich nacheinander Kraftstoffflüsse von 700 GpH und 600 GpH einstellen. Der Strom wird dann auf seinen Maximalwert von 900 mAMP gestellt und der Meßwert des resultierenden Flusses Q-Burner gespeichert.

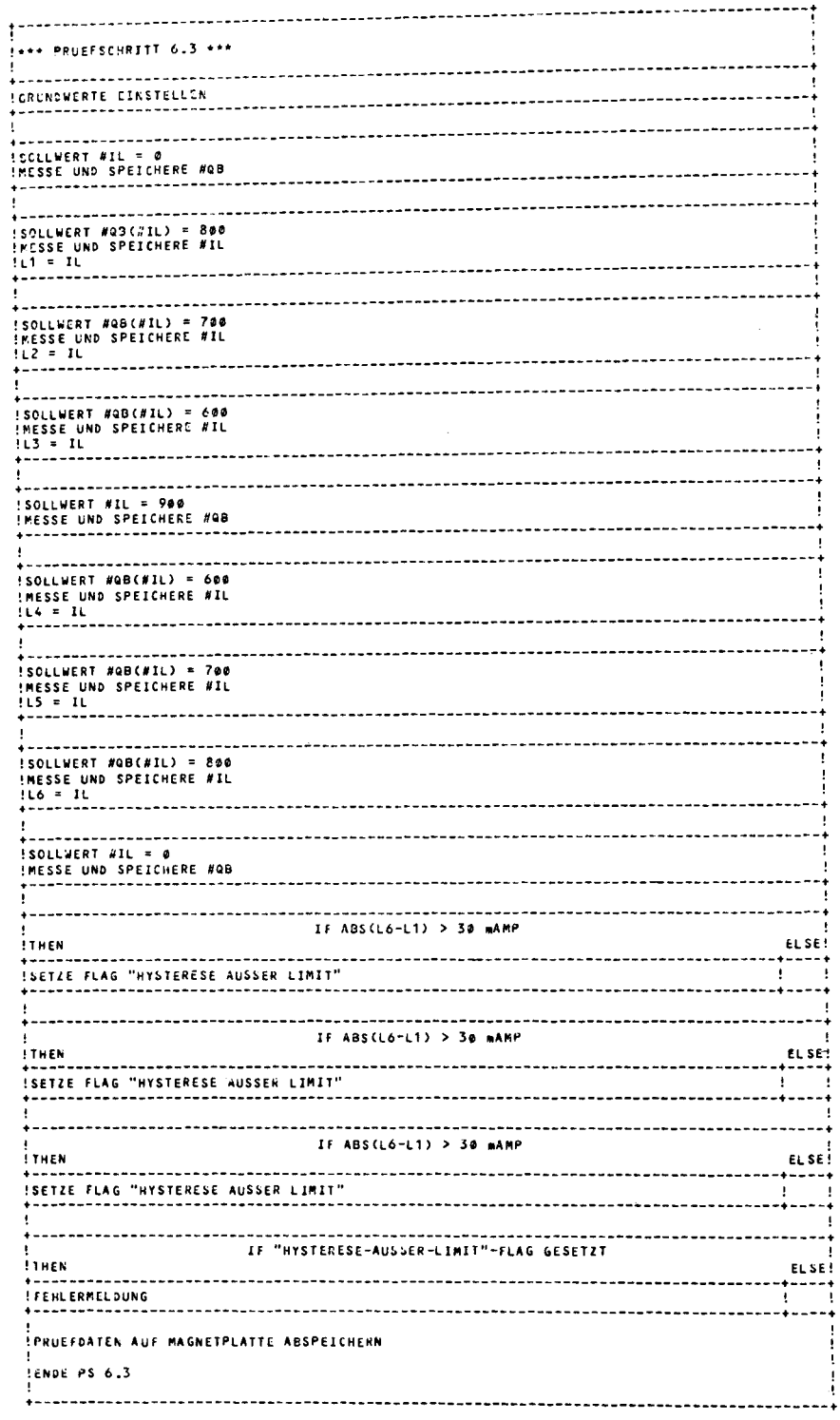
Im zweiten Teil der Prüfung wird der Steuerstrom I-LPC so gesenkt, daß sich stufenweise, jedoch in umgekehrter Reihenfolge, die gleichen Werte für den Fluß Q-Burner ergeben. Die Werte der für diese Einstellungen gemessenen Ströme werden gespeichert und nach Beendigung des Durchlaufs mit den Ergebnissen des ersten Teils der Prüfung verglichen. Ist die Differenz zwischen zwei korrespondierenden Meßwerten größer als die zulässige Hysterese von 30 mA, muß die Prüfung unterbrochen und der Kraftstoffregler mechanisch nachgearbeitet werden.



Struktogramm zu Prüfschritt 5.0



Struktogramm zu Prüfschritt 6.2



Struktogramm zu Prüfschritt 6.3

Programmlisting zu Prüfschritt 5.0

```
  5Ø INFO <1,Ø>
 1ØØ INFO <5,Ø> *** PRUEFSCHRITT 5.Ø ***
 1Ø1 INFO <5,1> VMO HOT SOAK / CYCLIC ENDURANCE
 11Ø *
 2ØØ * *** GRUNDWERTE ***
 2Ø1 *
 21Ø VAKUUMPUMPE AUS
 22Ø UMSCHALTUNG #P3
 23Ø DWV ATM
 24Ø STARTERFLOW ZU
 25Ø SOC1 EIN
 26Ø LPC LANE 1
 3ØØ SOLLWERT #QB(#NH)=Ø
 31Ø SOLLWERT #QB(#IL)=Ø
 315 SOLLWERT #KP(#P4)=Ø
 32Ø SOLLWERT #IL=Ø
 33Ø SOLLWERT #PP=Ø
 34Ø SOLLWERT #P3=Ø
 35Ø SOLLWERT #P4=Ø
 36Ø SOLLWERT #PT=15Ø
 37Ø SOLLWERT #NH=4ØØØ
 385 SOLLWERT #KP(#P4)=2Ø
 39Ø GRENZWERT #NH<35ØØ DANN 39Ø
 391 SOC1 AUS
 392 SOLLWERT #NH=771Ø
 395 DIM C(14)
 396 DIM B(14)
 397 *
 398 * **********
 399 *
 4ØØ *
 41Ø RECHNE I=Ø
 42Ø WARTE 1 SEC
 43Ø TOLERANZ #NH<77Ø5 ODER #NH>7715 DANN 41Ø
 44Ø RECHNE I=I+1
 45Ø WENN I=5 DANN 42Ø
 47Ø INFO <5,Ø> #P3-ANSCHLUSS ABBAUEN
 475 INFO <5,1> DANACH FORTSCHALTUNG DRUECKEN
 48Ø WARTE AUF FORTSCHALTUNG
 5ØØ *
 5Ø1 * **********
 5Ø2 *
1ØØØ *
1ØØ5 * **********
1Ø1Ø *
11ØØ INFO <5,1>.
12ØØ RECHNE X=2Ø
12Ø5 RECHNE Z=Ø
121Ø UNTERPROGRAMM 5ØØØ
13ØØ *
1301 * **********
1302 *
```

```
1310 RECHNE X=30
1315 RECHNE Z=1
1320 UNTERPROGRAMM 5000
1400 *
1401 * **********
1402 *
1410 RECHNE X=40
1415 RECHNE Z=2
1420 UNTERPROGRAMM 5000
1500 *
1501 * **********
1502 *
1510 RECHNE X=50
1515 RECHNE Z=3
1520 UNTERPROGRAMM 5000
1600 *
1601 * **********
1602 *
1610 RECHNE X=60
1615 RECHNE Z=4
1620 UNTERPROGRAMM 5000
1700 *
1701 * **********
1702 *
1710 RECHNE X=70
1715 RECHNE Z=5
1720 UNTERPROGRAMM 5000
1797 *
1798 * **********
1799 *
1900 INFO <5,0> #QB-TOLERANZ WIRD GEPRUEFT
1901 INFO <5,1>.
1920 UNTERPROGRAMM 50000
2000 *
2001 * **********
2002 *
2100 * DIE IN PS5.0 ERMITTELTEN PRUEFDATEN
2110 * MIT DENEN AUS PS4.3 VERGLEICHEN
2120 * DIFFERENZ MUSS KLEINER +-15 GPH SEIN
2130 *
2150 LESE 4.3
2200 FUER J=0 BIS 5
2210 DATA C(J)=#E(J*2+1)
2220 NAECHSTES J
2300 RECHNE FF=0
2310 FUER J=0 BIS 5
2320 WENN B(J*2+1)<C(J) DANN 2400
2330 WENN B(J*2+1)-C(J)>15 DANN 2500
2340 NAECHSTES J
2350 SPRINGE 3000
2400 WENN C(J)-B(J*2+1)>15 DANN 2500
2410 NAECHSTES J
2420 SPRINGE 3000
2500 RECHNE FF=1
2510 SPEICHERE #A(J*2+1)=?
2520 NAECHSTES J
2530 SPRINGE 3000
```

```
 3000 SCHREIBE 5.0
 3100 WENN FF=1 DANN 3500
 3110 INFO <5,0> ALLE TOLERANZEN O.K.
 3120 INFO <5,1> PS 5.0 ERFOLGREICH BEENDET
 3200 ENDE
 3297 *
 3298 * **********
 3299 *
 3500 *
 3520 INFO <5,0> #QB UEBERSCHREITET TOLERANZ
 3530 INFO <5,1> BITTE PRUEFPROTOKOLL BEACHTEN
 3600 ENDE
 3697 *
 3698 * **********
 3699 *
 5000 * * SUBR.* #KP4P EINSTELLEN
 5010 INFO <5,0>" #KP4P AUF ";X
 5015 SOLLWERT #KP(#P4)=X
 5020 RECHNE I=0
 5030 WARTE 1 SEC
 5040 TOLERANZ #KP<X-0.5 ODER #KP>X+0.5 DANN 5020
 5050 RECHNE I=I+1
 5060 WENN I<5 DANN 5030
 5100 * RETURN
 5197 *
 5198 * **********
 5199 *
10000 * * SUBR.: MESSWERTE #PV UND #QB ERFASSEN UND SPEICHERN
10100 RECHNE B(Z*2)=#PV
10110 SPEICHERE #A(Z*2)=B(Z*2)
10120 RECHNE B(Z*2+1)=#QB
10130 SPEICHERE #A(Z*2+1)=B(Z*2+1)
10200 ZURUECK
10297 *
10298 * **********
10299 *
50000 * SUBROUTINE "PRUEFSTAND HERUNTERFAHREN"
50005 *
50010 UMSCHALTUNG #P3
50020 SOLLWERT #NH=0
50030 SOLLWERT #P0=0
50040 SOLLWERT #P3=0
50050 SOLLWERT #P4=0
50060 SOLLWERT #PP=0
50080 SOLLWERT #PT=0
50085 SOLLWERT #QB(#NH)=0
50090 SOLLWERT #QB(#IL)=0
50092 SOLLWERT #IL=0
50095 SOLLWERT #KP(#P4)=0
50100 SOC1 AUS
50110 SOC2 AUS
50120 STARTERFLOW ZU
50130 LPC LANE 1
50140 VAKUUMPUMPE AUS
50150 DWV ATM
50160 CYCLE AUS
51000 ZURUECK
```

Programmlisting zu Prüfschritt 6.2

```
50 INFO <1,0>
100 INFO <5,0> *** PRUEFSCHRITT 6.2 ***
101 INFO <5,1> LPC INIT. / PULLDOWN LANE 1
110 *
200 * *** GRUNDWERTE ***
201 *
210 VAKUUMPUMPE AUS
220 UMSCHALTUNG #P3
230 DWV ATM
240 STARTERFLOW ZU
250 SOC1 EIN
260 LPC LANE 1
300 SOLLWERT #QB(#NH)=0
310 SOLLWERT #QB(#IL)=0
320 SOLLWERT #KP(#P4)=0
330 SOLLWERT #P0=0
340 SOLLWERT #PA=0
345 SOLLWERT #P3=75.7
350 SOLLWERT #NH=4000
355 SOLLWERT #P4=280
360 SOLLWERT #IL=0
370 SOLLWERT #PT=150
380 SOLLWERT #PP=0
385 GRENZWERT #NH<3500 DANN 385
390 SOC1 AUS
395 SOLLWERT #NH=7710
397 *
398 * **********
399 *
400 INFO <5,0> WARTE, BIS ALLE SOLLWERTE O.K.
410 RECHNE I=0
420 WARTE 1 SEC
430 TOLERANZ #PT<145 ODER #PT>155 DANN 410
440 TOLERANZ #P3<75.4 ODER #P3>76.2 DANN 410
450 TOLERANZ #P4<279 ODER #P4>281 DANN 410
460 TOLERANZ #NH<7708 ODER #NH>7712 DANN 410
470 RECHNE I=I+1
480 WENN I<3 DANN 420
490 INFO <5,0> SOLLWERTE SIND ERREICHT
500 *
501 * *********
502 *
510 UNTERPROGRAMM 520
515 SPRINGE 700
520 RECHNE X=350
530 UNTERPROGRAMM 10000
540 UNTERPROGRAMM 11000
560 RECHNE LO=710
570 RECHNE HI=730
580 UNTERPROGRAMM 12000
582 WENN OK<>0 DANN 650
585 INFO <5,1> MAIN-LPC-DATUM EINSTELLEN
```

```
590 WARTE AUF FORTSCHALTUNG
610 SPRINGE 580
650 *
660 SPEICHERE #A(1)=#QB
670 INFO <5,1>.
680 ZURUECK
700 *
701 * **********
702 *
710 RECHNE X=0
720 UNTERPROGRAMM 10000
730 UNTERPROGRAMM 11000
750 RECHNE LO=896
760 RECHNE HI=950
770 UNTERPROGRAMM 12000
780 WARTE AUF FORTSCHALTUNG
790 WENN OK<>0 DANN 850
795 INFO <5,1> ACU-ADJUSTER EINSTELLEN
797 WARTE AUF FORTSCHALTUNG
800 SPRINGE 750
850 *
860 SPEICHERE #A(0)=#QB
900 *
901 * **********
902 *
920 RECHNE X=900
930 UNTERPROGRAMM 10000
940 UNTERPROGRAMM 11000
950 RECHNE LO=458
960 RECHNE HI=486
970 UNTERPROGRAMM 12000
980 WARTE AUF FORTSCHALTUNG
990 WENN OK<>0 DANN 1050
995 INFO <5,1> DCU-ADJUSTER EINSTELLEN
997 WARTE AUF FORTSCHALTUNG
1000 SPRINGE 950
1050 *
1060 SPEICHERE #A(9)=#QB
1100 *
1101 * **********
1102 *
1110 INFO <5,0> TOLERANZ VON #QB BEI #IL = 350 MA
1112 INFO <5,1> NOCH EINMAL UEBERPRUEFEN
1115 SOLLWERT #IL=0
1120 WARTE 4 SEC
1130 UNTERPROGRAMM 520
1150 WARTE 2 SEC
1160 INFO <5,0> WEITERER ABLAUF AUTOMATISCH
1170 INFO <5,1>.
1180 WARTE 2 SEC
1185 *
1186 * **********
1187 *
1190 RECHNE X=100
1191 UNTERPROGRAMM 10000
1192 UNTERPROGRAMM 11000
1193 MESSE #QB
1194 SPEICHERE #A(2)=#QB
```

```
1197 *
1198 * **********
1199 *
1200 RECHNE X=0
1210 UNTERPROGRAMM 10000
1220 UNTERPROGRAMM 11000
1230 RECHNE #A1=#QB
1240 SPEICHERE #A(3)=#A1
1300 *
1301 * *********
1302 *
1310 RECHNE X=100
1320 UNTERPROGRAMM 10000
1330 UNTERPROGRAMM 11000
1335 RECHNE #B0=#QB
1340 SPEICHERE #A(4)=#B0
1350 TOLERANZ #QB<A1-5 ODER #QB>A1+5 DANN 5000
1400 *
1401 * *********
1402 *
1410 RECHNE X=180
1420 UNTERPROGRAMM 10000
1430 UNTERPROGRAMM 11000
1440 RECHNE #B1=#QB
1450 SPEICHERE #A(5)=#B1
1460 TOLERANZ #QB<B0-30 ODER #QB>B0 DANN 5000
1500 *
1501 * *********
1502 *
1510 RECHNE X=350
1520 UNTERPROGRAMM 10000
1530 UNTERPROGRAMM 11000
1540 RECHNE #C0=#QB
1550 SPEICHERE #A(6)=#C0
1600 *
1601 * **********
1602 *
1610 RECHNE X=400
1620 UNTERPROGRAMM 10000
1630 UNTERPROGRAMM 11000
1640 RECHNE #C1=#QB
1650 SPEICHERE #A(7)=#C1
1660 TOLERANZ #QB<C0-85 ODER #QB>C0-55 DANN 5000
1700 *
1701 * *********
1702 *
1710 RECHNE X=450
1720 UNTERPROGRAMM 10000
1730 UNTERPROGRAMM 11000
1740 RECHNE #C2=#QB
1750 SPEICHERE #A(8)=#C2
1760 TOLERANZ #QB<C0-155 ODER #QB>C0-125 DANN 5000
1800 *
1801 * *********
1802 *
1810 RECHNE X=900
1820 UNTERPROGRAMM 10000
1830 UNTERPROGRAMM 11000
```

```
 1840 RECHNE #D0=#QB
 1850 TOLERANZ #QB<458 ODER #QB>486 DANN 5000
 2000 *
 2001 *
 2002 *
 2003 * ***  ALLE TOLERANZEN IN ORDNUNG
 2004 *
 2005 *
 2006 SCHREIBE 6.2
 2010 * * PRUEFSTAND HERUNTERFAHREN *
 2020 INFO <5,0> * ALLE TOLERANZEN O.K. *
 2030 INFO <5,1> PS 6.2 ERFOLGREICH BEENDET
 2040 UNTERPROGRAMM 50000
 2300 ENDE
 5000 *
 5001 *
 5002 * *** #QB AUSSER TOLERANZ ***
 5004 *
 5005 *
 5010 INFO <5,0> #QB AUSSER TOLERANZ
 5030 WARTE AUF FORTSCHALTUNG
 5040 INFO <5,0> PRUEFSCHRITT WIEDERHOLEN
 5050 WARTE AUF FORTSCHALTUNG
 5060 SPRINGE 200
10000 *
10001 * *** SUBROUTINE:  #IL EINSTELLEN ***
10002 *
10002 *
10050 INFO <5,0>" STROM #I-LPC AUF ";X;"MAMP"
10060 INFO <5,1>.
10100 SOLLWERT #IL=X
10110 RECHNE I=0
10120 WARTE 1 SEC
10130 TOLERANZ #IL<X-1 ODER #IL>X+1 DANN 10110
10140 RECHNE I=I+1
10150 WENN I<3 DANN 10120
10200 ZURUECK
10997 *
10998 * **********
10999 *
11000 *
11001 * *** SUBROUTINE:  #P3 KONTROLLIEREN ***
11002 *
11010 RECHNE I1=0
11020 WARTE 1 SEC
11030 TOLERANZ #P3<75.4 ODER #P3>76.0 DANN 11010
11040 RECHNE I1=I1+1
11050 WENN I1<3 DANN 11020
11100 ZURUECK
11997 *
11998 * **********
11999 *
12000 *
12001 * *** SUBROUTINE: PRUEFE #QB
12002 *
12010 RECHNE OK=0
12030 WARTE 1 SEC
12040 GRENZWERT #QB<LO DANN 12200
12050 GRENZWERT #QB>HI DANN 12300
```

```
12060 INFO <5,0> #QB IN ORDNUNG
12065 MESSE #QB
12068 WARTE 3 SEC
12070 RECHNE OK=1
12080 ZURUECK
12200 INFO <5,0> #QB ZU NIEDRIG
12210 ZURUECK
12300 INFO <5,0> #QB ZU HOCH
12310 ZURUECK
12997 *
12998 * **********
12999 *
50000 * SUBROUTINE "PRUEFSTAND HERUNTERFAHREN"
50005 *
50010 UMSCHALTUNG #P3
50020 SOLLWERT #QB(#NH)=0
50030 SOLLWERT #QB(#IL)=0
50040 SOLLWERT #KP(#P4)=0
50050 SOLLWERT #P0=0
50055 SOLLWERT #PA=0
50060 SOLLWERT #P3=0
50065 SOLLWERT #P4=0
50070 SOLLWERT #PP=0
50075 SOLLWERT #PT=0
50080 SOLLWERT #IL=0
50085 SOLLWERT #NH=0
50100 SOC1 AUS
50110 SOC2 AUS
50120 STARTERFLOW ZU
50130 LPC LANE 1
50140 VAKUUMPUMPE AUS
50150 DWV ATM
50160 CYCLE AUS
51000 ZURUECK
```

Programmlisting zu Prüfschritt 6.3

```
  50 INFO <1,0>
 100 INFO <5,0> *** PRUEFSCHRITT 6.3 ***
 101 INFO <5,1> HYSTERESE LANE 1
 110 *
 200 * *** GRUNDWERTE ***
 201 *
 210 VAKUUMPUMPE AUS
 220 UMSCHALTUNG #P3
 230 DWV ATM
 240 STARTERFLOW ZU
 250 SOC1 EIN
 260 LPC LANE 1
 300 SOLLWERT #QB(#NH)=0
 310 SOLLWERT #QB(#IL)=0
 320 SOLLWERT #KP(#P4)=0
 330 SOLLWERT #P0=0
 340 SOLLWERT #PA=0
 345 SOLLWERT #P3=75.7
 355 SOLLWERT #P4=280
 360 SOLLWERT #IL=0
 370 SOLLWERT #PT=150
 380 SOLLWERT #PP=0
 385 SOLLWERT #NH=4000
 390 GRENZWERT #NH<3500 DANN 390
 391 SOC1 AUS
 395 SOLLWERT #NH=7710
 397 *
 398 * **********
 399 *
 400 INFO <5,0> WARTE, BIS ALLE SOLLWERTE O.K.
 410 RECHNE I=0
 420 WARTE 1 SEC
 430 TOLERANZ #PT<145 ODER #PT>155 DANN 410
 440 TOLERANZ #P3<75.2 ODER #P3>76.2 DANN 410
 450 TOLERANZ #P4<279 ODER #P4>281 DANN 410
 460 TOLERANZ #NH<7708 ODER #NH>7712 DANN 410
 470 RECHNE I=I+1
 480 WENN I<3 DANN 420
 489 INFO <5,1>.
 490 INFO <5,0> SOLLWERTE SIND ERREICHT
 495 WARTE 3 SEC
 500 *
 501 * *********
 502 *
 510 HALTE #P3
 520 INFO <5,0> #ILPC AUF 0 MAMP
 530 RECHNE X=0
 540 UNTERPROGRAMM 10000
 550 MESSE #QB
 560 SPEICHERE #A(0)=#QB
```

```
600 *
601 * **********
602 *
610 INFO <5,0> #I-LPC ERHOEHEN, BIS
611 INFO <5,1> #QB AUF 800 GPH
620 RECHNE Y=800
630 UNTERPROGRAMM 12000
640 RECHNE #L1=#IL
650 SPEICHERE #A(2)=#L1
700 *
701 * **********
702 *
710 INFO <5,1> #QB AUF 700 GPH
720 RECHNE Y=700
730 UNTERPROGRAMM 12000
740 RECHNE #L2=#IL
750 SPEICHERE #A(5)=#L2
800 *
801 * **********
802 *
810 INFO <5,1> #QB AUF 600 GPH
820 RECHNE Y=600
830 UNTERPROGRAMM 12000
840 RECHNE #L3=#IL
850 SPEICHERE #A(8)=#L3
900 *
901 * **********
902 *
910 INFO <5,0> #ILPC AUF 900 MAMP
915 INFO <5,1>.
920 RECHNE X=900
930 UNTERPROGRAMM 10000
940 MESSE #QB
950 SPEICHERE #A(11)=#QB
1000 *
1001 * **********
1002 *
1010 SPEICHERE #A(12)=#QB
1100 *
1101 * **********
1102 *
1105 INFO <5,0> #I-LPC VERRINGERN, BIS
1110 INFO <5,1> #QB AUF 600 GPH
1120 RECHNE Y=600
1130 UNTERPROGRAMM 12000
1140 RECHNE #L4=#IL
1150 SPEICHERE #A(10)=#L4
1200 *
1201 * **********
1202 *
1210 INFO <5,1> #QB AUF 700 GPH
1220 RECHNE Y=700
1230 UNTERPROGRAMM 12000
1240 RECHNE #L5=#IL
1250 SPEICHERE #A(7)=#L5
1300 *
1301 * **********
```

```
 1302 *
 1310 INFO <5,1> #QB AUF 800 GPH
 1320 RECHNE Y=800
 1330 UNTERPROGRAMM 12000
 1340 RECHNE #L6=#IL
 1350 SPEICHERE #A(4)=#L6
 1400 *
 1401 * **********
 1402 *
 1410 INFO <5,0> #ILPC AUF 0 MAMP
 1415 INFO <5,1>.
 1420 RECHNE X=0
 1430 UNTERPROGRAMM 10000
 1440 MESSE #QB
 1450 SPEICHERE #A(1)=#QB
 2000 *
 2001 * **********
 2002 *
 2004 * HYSTERESE DES STORMES BERECHNEN
 2010 RECHNE D1=L1-L6
 2020 RECHNE D2=L2-L5
 2030 RECHNE D3=L3-L4
 2040 SPEICHERE #A(3)=D1
 2050 SPEICHERE #A(6)=D2
 2060 SPEICHERE #A(9)=D3
 2100 * PRUEFE, OB STROM-HYST. > 30 MAMP
 2110 WENN D1>30 DANN 2310
 2120 WENN D2>30 DANN 2320
 2130 WENN D3>30 DANN 2330
 2200 * ALLE  HYST. IM LIMIT
 2210 SCHREIBE 6.3
 2220 INFO <5,0> HYSTERESE O.K.
 2225 INFO <5,1> PS 6.3 ERFOLGREICH BEENDET
 2230 UNTERPROGRAMM 50000
 2240 ENDE
 2295 *
 2296 * **********
 2297 *
 2300 INFO <5,0> HYSTERESE ZU GROSS
 2310 SPEICHERE #A(3)=?
 2315 SPRINGE 2340
 2320 SPEICHERE #A(6)=?
 2325 SPRINGE 2340
 2330 SPEICHERE #A(9)=?
 2340 INFO <5,0> HYSTERESE #I-LPC ZU GROSS
 2345 INFO <5,1> MAIN-LPC MUSS ABGEBAUT WERDEN
 2347 SCHREIBE 6.3
 2350 UNTERPROGRAMM 50000
 2360 ENDE
 2395 *
 2396 * **********
 2397 *
10000 *
10001 * *** SUBROUTINE:  #IL EINSTELLEN ***
10002 *
10100 SOLLWERT #IL=X
10110 RECHNE I=0
10120 WARTE 1 SEC
```

```
10130 TOLERANZ #IL<X-1 ODER #IL>X+1 DANN 10110
10140 RECHNE I=I+1
10150 WENN I<3 DANN 10120
10200 ZURUECK
12000 *
12001 * *** SUBROUTINE:  #QB EINSTELLEN ***
12002 *
12010 SOLLWERT #QB(#IL)=Y
12020 RECHNE I2=0
12030 WARTE 1 SEC
12040 TOLERANZ #QB<Y-1 ODER #QB>Y+1 DANN 12020
12050 RECHNE I2=I2+1
12060 WENN I2<3 DANN 12030
12070 ZURUECK
12095 *
12097 * *********
12098 *
50000 * SUBROUTINE "PRUEFSTAND HERUNTERFAHREN"
50005 *
50010 UMSCHALTUNG #P3
50020 SOLLWERT #QB(#NH)=0
50030 SOLLWERT #QB(#IL)=0
50040 SOLLWERT #KP(#P4)=0
50050 SOLLWERT #P0=0
50055 SOLLWERT #PA=0
50060 SOLLWERT #P3=0
50065 SOLLWERT #P4=0
50070 SOLLWERT #PP=0
50075 SOLLWERT #PT=0
50080 SOLLWERT #IL=0
50085 SOLLWERT #NH=0
50100 SOC1 AUS
50110 SOC2 AUS
50120 STARTERFLOW ZU
50130 LPC LANE 1
50140 VAKUUMPUMPE AUS
50150 DWV ATM
50160 CYCLE AUS
51000 ZURUECK
```

Blatt: 2 von 12	FLOW CONTROL - ENGINE FUEL	Serien-Nr.: 0502

5.0 V.M.O. HOT SOAK AND CYCLIC ENDURANCE TEST

DREH-ZAHL 1/MIN	KP4P PSIA	DIFFERENZ-DRUCK PSI VMO	DURCHFLUSS GPH
8116	110	————	45
8116	U-110	50x Wechseln, 5x pro minute	

7710	20	104	A+-15	165.7
7710	30	109	B+-15	266.8
7710	40	109	C+-15	388
7710	50	109	D+-15	503
7710	60	110	E+-15	620
7710	70	103	F+-15	732

6.2. L.P.C. INITIATION AND RATE OF PULLDOWN LANE 1

DREHZAHL 1/MIN	7705	P4 PSIA	280.0	P3 PSIA	75.7

DURCHFLUSS GPH			STROM mA	
896	943	950	0	
710	727	730	350	
896	932	950	100	
——	942	——	0	A
A-5	939	A+5	100	B
B-30	925	B	180	
——	721	——	350	C
C-85	646	C-55	400	
C-185	573	C-125	450	
458	482	486	900	

6.3 HYSTERESIS LANE 1 < = Erhoehen
> = Verringern

STROM mA <	DURCHFLUSS GPH	STROM DELTA MAX 30 mA	DURCHFLUSS GPH	STROM mA >
0	941	—	939	0
316	800	10	800	306
394	700	11	700	383
485	600	13	600	472
900	546	—	546	900

Prüfprotokoll der Prüfschritte 5.0, 6.2 und 6.3

B. Bildschirm-Steuerzeichen

Liste der zugelassenen Bildschirm-Steuerzeichen für Statements INFO, GRAPHIK und WARTE AUF EINGABE

1, 0	Bildschirm löschen, Analoganzeige ein	I G
1, 1	Analoganzeige aus, Bildschirm löschen	I G
1, 2	Bildschirm löschen	I G
2, 0	erste Textzeile Bildmitte	I
2, 1	zweite Textzeile Bildmitte	I
3, 0	Analoganzeige aus, Bildschirm löschen, Eingabemaske einblenden	I G
3, 1	Eingabe in Datenfeld der Eingabemaske	W
5, 0	Erste Textzeile unten rechts	I
5, 1	Zweite Textzeile unten rechts	I

C. Prozeßvariablen-Namen

Liste der in der vorliegenden Implementation zugelassenen Prozeßvariablen-Namen:

Analog-Nur-Meßwerte

Prozeßvariablenname	Funktion
≠ KP	Luftdruck KP4P
≠ PN	Kraftstoff P-Burner
≠ PS	Luftdruck PDSVMO

Digitale Meßwerte

Prozeßvariablenname	Funktion
≠ QB	Kraftstoffffluß-Q-Burner
≠ TE	Kraftstofftemperatur

Errechnete Prozeßgrößen (Differenzdrücke)

Prozeßvariablenname	Funktion
≠ PR	Differenzdruck PDPRV=RDSVMO - P-Burner
≠ PV	Differenzdruck PDVMO=PPUMP - PDSVMO

Einfach-Regelkreise

Prozeßvariablenname	Funktion
≠ PØ	Luftdruck PØ
≠ P3	Luftdruck P3
≠ P4	Luftdruck P4
≠ PT	Kraftstoffdruck P-Boost
≠ PP	Kraftstoffdruck P-PUMP
≠ IL	Strom I-LPC
≠ NH	Pumpendrehzahl NH
≠ PA	Luftdruck P3A (Bereich unter Atmosphäre)

Folgeregelkreise

Prozeßvariablenname	Funktion
≠ KP (≠ P4)	Luftdruck KP4P, über P4 beeinflußt
≠ QB (≠ NH)	Kraftstoffffluß, über Q-Burner beeinflußt
≠ QB (≠ IL)	Kraftstoffffluß, über I-LPC beeinflußt

Die Autoren möchten Herrn Dr. Müller vom Zentrallabor für Elektronik der KFA- Jülich und Herrn Hendrix von der Fa. Pierburg Luftfahrtgeräte Union für die tatkräftige Unterstützung bei dieser Arbeit danken. Insbesondere Herr Hendrix und die zuständigen Mitarbeiter im Prüfstands- und Versuchsbereich sowie in der Qualitätssicherung haben durch wertvolle Hinweise und Diskussionen über die meßtechnischen und anwendungsorientierten Probleme entscheidend zum Gelingen beigetragen.

Herrn Peine von der Fa. Pierburg gilt unser besonderer Dank für seinen Einsatz beim Aufbau des gesamten Hardwaresystems.